STATA LASSO REFERENCE MANUAL
RELEASE 19

A Stata Press Publication
StataCorp LLC
College Station, Texas

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845

ISBN-10: 1-59718-433-0
ISBN-13: 978-1-59718-433-5

The suggested citation for this software is

StataCorp. 2025. *Stata 19*. Statistical software. StataCorp LLC.

The suggested citation for this manual is

StataCorp. 2025. *Stata 19 Lasso Reference Manual*. College Station, TX: Stata Press.

www.stata.com

Contents

Cross-referencing the documentation

When reading this manual, you will find references to other Stata manuals, for example, **[U] 27 Overview of Stata estimation commands**; **[R] regress**; and **[D] reshape**. The first example is a reference to chapter 27, *Overview of Stata estimation commands*, in the *User's Guide*; the second is a reference to the `regress` entry in the *Base Reference Manual*; and the third is a reference to the `reshape` entry in the *Data Management Reference Manual*.

All the manuals in the Stata Documentation have a shorthand notation:

[GSM]	*Getting Started with Stata for Mac*
[GSU]	*Getting Started with Stata for Unix*
[GSW]	*Getting Started with Stata for Windows*
[U]	*Stata User's Guide*
[R]	*Stata Base Reference Manual*
[ADAPT]	*Stata Adaptive Designs: Group Sequential Trials Reference Manual*
[BAYES]	*Stata Bayesian Analysis Reference Manual*
[BMA]	*Stata Bayesian Model Averaging Reference Manual*
[CAUSAL]	*Stata Causal Inference and Treatment-Effects Estimation Reference Manual*
[CM]	*Stata Choice Models Reference Manual*
[D]	*Stata Data Management Reference Manual*
[DSGE]	*Stata Dynamic Stochastic General Equilibrium Models Reference Manual*
[ERM]	*Stata Extended Regression Models Reference Manual*
[FMM]	*Stata Finite Mixture Models Reference Manual*
[FN]	*Stata Functions Reference Manual*
[G]	*Stata Graphics Reference Manual*
[H2OML]	*Machine Learning in Stata Using H2O: Ensemble Decision Trees Reference Manual*
[IRT]	*Stata Item Response Theory Reference Manual*
[LASSO]	*Stata Lasso Reference Manual*
[XT]	*Stata Longitudinal-Data/Panel-Data Reference Manual*
[META]	*Stata Meta-Analysis Reference Manual*
[ME]	*Stata Multilevel Mixed-Effects Reference Manual*
[MI]	*Stata Multiple-Imputation Reference Manual*
[MV]	*Stata Multivariate Statistics Reference Manual*
[PSS]	*Stata Power, Precision, and Sample-Size Reference Manual*
[P]	*Stata Programming Reference Manual*
[RPT]	*Stata Reporting Reference Manual*
[SP]	*Stata Spatial Autoregressive Models Reference Manual*
[SEM]	*Stata Structural Equation Modeling Reference Manual*
[SVY]	*Stata Survey Data Reference Manual*
[ST]	*Stata Survival Analysis Reference Manual*
[TABLES]	*Stata Customizable Tables and Collected Results Reference Manual*
[TS]	*Stata Time-Series Reference Manual*
[I]	*Stata Index*
[M]	*Mata Reference Manual*

Lasso intro — Introduction to lasso

Description

Lasso was originally an acronym for "least absolute shrinkage and selection operator". Today, lasso is considered a word and not an acronym.

Lasso is used for prediction, for model selection, and as a component of estimators to perform inference.

Lasso, elastic net, and square-root lasso are designed for model selection and prediction. Stata's `lasso`, `elasticnet`, and `sqrtlasso` commands implement these methods. `lasso` and `elasticnet` fit continuous, binary, count, and failure-time outcomes, while `sqrtlasso` fits continuous outcomes.

Stata also provides lasso commands for inference. They use lassos to select control variables that appear in the model, and they estimate coefficients and standard errors for a specified subset of covariates.

Stata's lasso inference commands implement methods known as double selection, partialing out, and cross-fit partialing out. With each of these methods, linear, logistic, or Poisson regression can be used to model a continuous, binary, or count outcome. Partialing out and cross-fit partialing out also allow for endogenous covariates in linear models.

Stata also provides a specialized lasso inference command for estimating treatment effects while using lassos to select control variables. `telasso` estimates the average treatment effect (ATE), average treatment effect on the treated (ATET), or potential-outcome means (POMs); see [CAUSAL] **telasso**.

This entry provides an overview of lasso for prediction, model selection, and inference and an introduction to Stata's suite of lasso commands.

Remarks and examples

Remarks are presented under the following headings:

Summary of Stata's lasso and elastic-net features

For those of you who are already experts on lasso, here is an overview full of buzz words without explanations.

Stata provides three estimation commands for prediction and model selection.

`lasso`	fits linear, logit, probit, Poisson, and Cox proportional hazards models. The final model is selected using cross-validation (CV), adaptive lasso, plugin estimators, or the Bayesian information criterion (BIC) function.
`elasticnet`	also fits linear, logit, probit, Poisson, and Cox proportional hazards models. `elasticnet` uses CV or the BIC function to select models.
`sqrtlasso`	fits square-root lassos for linear models. The final model is selected using CV, plugin estimators, or the BIC function.

After fitting a model, you obtain out-of-sample predictions by loading another dataset and typing

```
. predict newvarname
```

Stata provides another 11 lasso commands for use in inference. These 11 commands are arranged in three groups.

ds commands	perform double-selection lasso: `dsregress` for linear models, `dslogit` for logit models, and `dspoisson` for Poisson models.
po commands	perform partialing-out lasso: `poregress` for linear models, `pologit` for logit models, `popoisson` for Poisson models, and `poivregress` for linear models with endogenous covariates.
xpo commands	perform cross-fit partialing-out lasso, also known as double machine learning: `xporegress` for linear models, `xpologit` for logit models, `xpopoisson` for Poisson models, and `xpoivregress` for linear models with endogenous covariates.

Stata provides one additional lasso command for use in inference when your objective is to estimate treatment effects.

`telasso`	fits linear, logit, probit, and Poisson models and estimates the ATE, ATET, or POMs.

Stata provides two preestimation commands that help you prepare your data and specify covariates.

`splitsample`	divides your sample into k random subsamples. Use it for producing subsamples for training, validation, and prediction.
`vl`	creates named lists of variables to be included in `lasso`. Sometimes, you will want to specify many potential covariates without typing every variable name. `vl` creates named variable lists that can be used as command arguments. Lists can contain hundreds or thousands of variable names.

Stata provides eight postestimation commands that help you evaluate the selected model:

`bicplot`	graphs the BIC function.
`cvplot`	graphs the CV function.
`coefpath`	graphs coefficient paths.
`lassoknots`	displays a knot table for covariates as they enter or leave the model and measures of fit.
`lassogof`	reports fit statistics that help you evaluate the predictive ability of a model. It does this for one model or for multiple models in the same table.
`lassocoef`	lists the selected variables in the model. It does this for one model or for multiple models in the same table.
`lassoselect`	selects a different model from the one chosen by the estimation command.
`lassoinfo`	reports lasso information such as the dependent variable, selection method, and number of nonzero coefficients for one or more models.

What is lasso?

Lasso is a method for selecting and fitting covariates that appear in a model. The `lasso` command can fit linear, logit, probit, Poisson, and Cox proportional hazards models. Let's consider a linear model, a model of `y` on `x1`, `x2`, …, `x`*p*. You would ordinarily fit this model by typing

```
. regress y x1 x2 ... xp
```

Now assume that you are uncertain which variables (covariates) belong in the model, although you are certain that some of them do and the number of them is small relative to the number of observations in your dataset, N. In that case, you can type

```
. lasso linear y x1 x2 ... xp
```

You can specify hundreds or even thousands of covariates. You can even specify more covariates than there are observations in your data! The covariates you specify are the potential covariates from which lasso selects.

Lasso is used in three ways:

1. Lasso is used for prediction.
2. Lasso is used for model selection.
3. Lasso is used for inference.

By prediction, we mean predicting the value of an outcome conditional on a large set of potential regressors. And we mean predicting the outcome both in and out of sample.

By model selection, we mean selecting a set of variables that predicts the outcome well. We do not mean selecting variables in the true model or placing a scientific interpretation on the coefficients. Instead, we mean selecting variables that correlate well with the outcome in one dataset and testing whether those same variables predict the outcome well in other datasets.

By inference, we mean inference for interpreting and giving meaning to the coefficients of the fitted model. Inference is concerned with estimating effects of variables in the true model and estimating standard errors, confidence intervals, p-values, and the like.

Lasso for prediction

Lasso was invented by Tibshirani (1996) and has been commonly used in building models for prediction. Hastie, Tibshirani, and Wainwright (2015) provide an excellent introduction to the mechanics of the lasso and to the lasso as a tool for prediction. See Bühlmann and van de Geer (2011) for more technical discussion and clear discussion of the properties of lasso under different assumptions. See Cameron and Trivedi (2022, chap. 28) for an introduction to lasso for prediction and for inference with examples using Stata.

Lasso does not necessarily select the covariates that appear in the true model, but it does select a set of variables that are correlated with them. If lasso selects potential covariate `x47`, that means `x47` belongs in the model or is correlated with variables that belong in the model. If lasso omits potential covariate `x52`, that means `x52` does not belong in the model or belongs but is correlated with covariates that were already selected. Because we are interested only in prediction, we are not concerned with the exact variables selected, only that they are useful for prediction.

The model lasso selects is suitable for making predictions in samples outside the one you used for estimation. Everyone knows about the danger of overfitting. Fit a model on one set of data and include too many variables, and the result will exploit features randomly unique to those data that will not be replicated in other data.

"Oh," you may be thinking, "you mean that I can split my data into an estimation sample and a hold-out sample, and after fitting, I can evaluate the model in the hold-out sample." That is not what we mean, although you can do this, and it is sometimes a good idea to do so. We mean that lasso works to avoid the problem of overfitting by minimizing an estimate of the out-of-sample prediction error.

How lasso for prediction works

Lasso finds a solution for

$$\mathbf{y} = \beta_1 \mathbf{x}_1 + \beta_2 \mathbf{x}_2 + \cdots + \beta_p \mathbf{x}_p + \boldsymbol{\epsilon}$$

by minimizing the prediction error subject to the constraint that the model is not too complex—that is, it is sparse. Lasso measures complexity by the sum of the absolute values of $\beta_1, \beta_2, \dots, \beta_p$. The solution is obtained by minimizing

$$\frac{1}{2N}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}')'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}') + \lambda \sum_{j=1}^{p} |\beta_j| \tag{1}$$

The first term, $(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}')'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}')$, is the in-sample prediction error. It is the same value that least squares minimizes.

The second term, $\lambda \sum_j |\beta_j|$, is a penalty that increases in value the more complex the model. It is this term that causes lasso to omit variables. They are omitted because of the nondifferentiable kinks in the $\sum_j |\beta_j|$ absolute value terms. Had the kinks not been present—think of squared complexity terms rather than absolute value—none of the coefficients would be exactly zero. The kinks cause some coefficients to be zero.

If you minimized (1) with respect to the β_j's and λ, the solution would be $\lambda = 0$. That would set the penalty to zero. $\lambda = 0$ corresponds to a model with maximum complexity.

Lasso proceeds differently. It minimizes (1) for given values of λ. Lasso then chooses one of those solutions as best based on another criterion, such as an estimate of the out-of-sample prediction error.

When we use lasso for prediction, we must assume the unknown true model contains few variables relative to the number of observations, N. This is known as the sparsity assumption. How many true variables are allowed for a given N? We can tell you that the number cannot be greater than something proportional to $\sqrt{N}/\ln q$, where $q = \max\{N, p\}$ and p is the number of potential variables. We cannot, however, say what the constant of proportionality is. That this upper bound decreases with q can be viewed as the cost of performing covariate selection.

Lasso provides various ways of selecting λ: through CV, adaptive lasso, a plugin estimator, or minimizing the Bayesian information criterion (BIC) function. CV selects the λ that minimizes an estimate of the out-of-sample prediction error. Adaptive lasso performs multiple lassos, each with CV. After each lasso, variables with zero coefficients are removed, and remaining variables are given penalty weights designed to drive small coefficients to zero. Thus, adaptive lasso typically selects fewer covariates than CV.

The plugin method was designed to achieve an optimal sparsity rate. It tends to select a larger λ than CV and, therefore, fewer covariates in the final model. The number of covariates selected by minimizing BIC typically lies between the number selected by CV and the number selected by the plugin method; however, BIC tends to be more similar to the number selected by the plugin method. Furthermore, BIC does not require a complex derivation as does the plugin, so like CV, it can be applied in a more general context. See [LASSO] **lasso** and [LASSO] **lasso fitting** for more information on the methods of selecting λ, their differences, and how you can control the selection process.

Stata commands for prediction

We described the linear lasso model in the last section, but the concepts we have discussed apply to models for binary and count outcomes as well.

To fit a linear lasso model, we might type

```
. lasso linear y x1-x500
```

and `lasso` will select a subset of variables from `x1` to `x500` that can be used in prediction.

If we have a binary outcome, we could instead fit a logit model by typing

```
. lasso logit y x1-x500
```

or a probit model by typing

```
. lasso probit y x1-x500
```

For a count outcome, we could fit a Poisson model by typing

```
. lasso poisson y x1-x500
```

For failure-time data that has been `stset`, we could fit a Cox proportional hazards model by typing

```
. lasso cox x1-x500
```

After any of these `lasso` commands, we can use `predict` to obtain predictions of `y`.

For examples demonstrating how to use the `lasso` command to fit models suitable for prediction, see *Remarks and examples* in [LASSO] **lasso** and also see [LASSO] **lasso examples**.

Stata also has commands for fitting elastic nets and square-root lassos for prediction. See [LASSO] **elasticnet** and [LASSO] **sqrtlasso** for more information and examples.

Lasso for model selection

Model selection is an overloaded term that implies different things in different disciplines. To some, it implies finding a true model or data-generating process. To some, it implies less. Here model selection means finding a model that fits the data, not finding a model that allows for interpreting estimated coefficients as effects. If this is your interest, see *Lasso for inference* below.

Lasso for model selection builds on lasso for prediction. In fact, the same lasso methods are used in both cases. However, the goal of the analysis is different.

Model selection uses lasso to select variables in one dataset and then fits models using the selected variables in other datasets. For example, consider finding genes correlated with an outcome in microarray data. One approach starts with lasso. Researchers use it as a sieve to select the important predictors. They go on to test whether those predictors (genetic markers) work in other datasets. Note that these researchers are not giving scientific meaning to the estimated coefficients. They are looking only for markers that correlate well with an outcome.

We can perform these types of tests because our interest lies in the selected model rather than the true coefficients of the data-generating process (DGP), sometimes called the data-generating mechanism (DGM). Interpretation is conditional on the selected model and cannot be interpreted as causal. See, for instance, Lee et al. (2016). As Berk et al. (2013) put it, the goal is "… to merely describe association between predictor and response variables; no data generating or causal claims are implied."

Lasso for inference

When we use lasso for inference, we are interested in interpreting the estimated coefficients. We are also interested in standard errors, hypothesis tests, confidence intervals, comparisons across levels, and the like. We want to interpret the results in the same way we interpret results from standard regression models.

Why do we need special lasso methods for inference?

It may be tempting to use lasso to select covariates and then use `regress` (or `logit`, `probit`, or `poisson`) to fit a model with the selected covariates. The results from the regression provide estimated coefficients and standard errors, confidence intervals, and p-values.

This approach does not work. Why?

Consider fitting a classic regression model. The standard error for a coefficient tells us about the distribution of the coefficient in repeated sampling. The 95% confidence interval includes the true value of the coefficient in 95% of repeated samples. Although we do not actually have repeated samples from the population, the standard errors allow us to account for sample-to-sample variability when making inferences.

If we use lasso to select covariates and then use `regress` to fit a model with only the selected covariates, the results will be problematic for use in inference for a few reasons.

First, when we use lasso, or any variable-selection method, we introduce a new source of variability. If we actually drew repeated samples from a population and used `lasso` to select covariates on each one, different covariates would be selected in each dataset. However, we have selected covariates using only a single sample. The standard errors reported by `regress` do not account for the sample-to-sample variability in the variable selection.

Second, lasso tends to omit covariates with small coefficients. This problem arises because lasso minimizes prediction error subject to the constraint that the model is not too complex, and lasso measures complexity by the sum of the absolute values of the coefficients. Covariates with small coefficients tend to be entrapped by the constraint. Small coefficients of covariates that belong in the model look just like small coefficients of variables that do not. Mistakenly omitted covariates, even those with small coefficients, can bias other coefficients. That bias is not solely a function of the size of the coefficient. See, for instance, Leeb and Pötscher (2005, 2006, 2008) and Belloni, Chernozhukov, and Hansen (2014).

And then there are more mundane reasons the selected variables can differ from the true variables. Imagine that you have fit a lasso model. You look at the results and observe that region-of-country covariates are included. You are surprised because you can think of no reason why they should be and wonder whether you are about to make an interesting discovery. You look at the lasso results in hopes of finding an explanation. You discover that income was excluded despite your expectations to the contrary. You find that age and education were included, but that does not surprise you. But region, age, and education are predictors of income. That could be the entire reason why region covariates were included. They were included only because income was excluded. Or it could be something deeper.

In general, the variables selected by lasso do not even converge to the ones in the true model as the number of observations goes to infinity. Lasso tends to omit covariates that have small coefficients in favor of irrelevant ones (variables not in the true model) that are correlated with the error term.

For these reasons, we must use lasso-based methods that are designed specifically for inference when we want to interpret coefficients.

Methods of lasso for inference

With lasso inferential methods, researchers wish to interpret the covariates selected by lasso in the context of the DGP. They apply causal interpretations to the results. This approach accounts for the fact that lasso does not select the true model with probability one, and it accounts for the errors that arise in model selection. To achieve this DGP causal interpretation, you must perform the selection process with resampling. Thus, more than split sampling is needed to obtain consistent standard errors. See Belloni, Chernozhukov, and Hansen (2014) for an excellent introduction to using lasso to perform inference and make causal interpretations.

When your interest is in the underlying DGP, there are various ways of using lasso to estimate the effects of a few covariates that you have chosen a priori. These methods may be used when you know there are more covariates in the model and that they are sparse (relative to N). These methods apply, for instance, to performing inference about the effect of smoking on a health outcome when you know that lots of other variables potentially affect the outcome but do not know which ones.

The double-selection, partialing-out, and cross-fit partialing-out lassos provided by Stata can handle such problems.

Other methods for inference have been discussed in the literature. For instance, see van de Geer et al. (2014), Javanmard and Montanari (2014), and Zhang and Zhang (2014). The methods developed there are not implemented in Stata. While they have some appealing theoretical properties, they have not yet been much used in applied work.

Stata commands for inference

For inference, multistage extensions of lasso provide standard errors for a subset of variables that you specify. Imagine that you wish to estimate the coefficients for `d1` and `d2` in the model that includes other covariates:

```
. regress y d1 d2 x1-x500
```

Covariates `x1` through `x500` are control variables, some of which you need to include to obtain valid results for `d1` and `d2`. Suppose your data contain 1,000 observations.

If all 500 covariates belong in the model, there is no way to proceed. Get more data. If only a small subset of them is required and you simply do not know which they are, there is a lasso-based solution. Type

```
. dsregress y d1 d2, controls(x1-x500)
```

Coefficients and standard errors for `d1` and `d2` will be reported. `dsregress` will use lasso to select from the 500 covariates and do that in a way that is robust to the model-selection mistakes that lasso makes because of sampling variability. There is no a priori limit on the number of **d** variables you can specify. But more variables mean more computation time. Time is roughly proportional to the number of **d** variables.

Stata provides three methods to fit these types of inferential models. They are

1. the ds double-selection commands: `dsregress`, `dslogit`, and `dspoisson`.
2. the po partialing-out commands: `poregress`, `pologit`, `popoisson`, and `poivregress`.
3. the `xpo` cross-fit partialing-out commands, also known as double machine learning: `xporegress`, `xpologit`, `xpopoisson`, and `xpoivregress`.

All three methods require a sparsity assumption. As with lasso for prediction, these methods of lasso for inference rely on the assumption that the number of nonzero coefficients in the true model is small relative to the number of observations and that the coefficients are large enough relative to error variance to be selected by the lasso.

ds and po are asymptotically equivalent. poivregress can handle d1 and d2 being endogenous in linear models. It does this using instrumental variables. You specify a set of potential instruments, and lasso will select from among them. You can have many potential control variables and many potential instruments; the number of each can be greater than N.

xpo is the most computationally intensive of the three methods. It is also generally viewed as superior to ds and po because it allows a weaker definition of sparsity. The sparsity bound for the ds and po methods grows in proportion to $\sqrt{N}$, while the sparsity bound for the xpo method grows in proportion to N.

For information on the assumptions and how the ds, po, and xpo commands work, see [LASSO] **Lasso inference intro** and [LASSO] **Inference requirements**.

For examples of fitting lasso inferential models, see [LASSO] **Inference examples**.

Where to learn more

After reading this intro, you may want to learn more about lasso for prediction and model selection, lasso for inference, and syntax for lasso commands, or you may just want to see some examples. Here we provide a guide to the entries in this manual that you may want to read next.

If you are interested in lasso for prediction or model selection, you may want to go directly to the syntax and examples demonstrating lasso, square-root lasso, and elastic net in

[LASSO] **lasso**	Lasso for prediction and model selection
[LASSO] **sqrtlasso**	Square-root lasso for prediction and model selection
[LASSO] **elasticnet**	Elastic net for prediction and model selection
[LASSO] **lasso examples**	Examples of lasso for prediction

If you are interested in lasso for inference, you can read more about the concepts, methods, and corresponding Stata commands in

[LASSO] **Lasso inference intro**	Introduction to inferential lasso models

If you want to see syntax for one of the lassos for inference commands, see

[LASSO] **dsregress**	Double-selection lasso linear regression
[LASSO] **dslogit**	Double-selection lasso logistic regression
[LASSO] **dspoisson**	Double-selection lasso Poisson regression
[LASSO] **poregress**	Partialing-out lasso linear regression
[LASSO] **pologit**	Partialing-out lasso logistic regression
[LASSO] **popoisson**	Partialing-out lasso Poisson regression
[LASSO] **poivregress**	Partialing-out lasso instrumental-variables regression
[LASSO] **xporegress**	Cross-fit partialing-out lasso linear regression
[LASSO] **xpologit**	Cross-fit partialing-out lasso logistic regression
[LASSO] **xpopoisson**	Cross-fit partialing-out lasso Poisson regression
[LASSO] **xpoivregress**	Cross-fit partialing-out lasso instrumental-variables regression
[LASSO] **lasso options**	Lasso options for inferential models

You might instead want to start with worked examples that demonstrate the lasso inference commands.

[LASSO] **Inference examples**	Examples and workflow for inference

If your inference involves estimating treatment effects, you can read about the lasso inference command that estimates the ATE, ATET, or POMs at

[CAUSAL] **telasso** Treatment-effects estimation using lasso

Whether you are using lasso for prediction or for inference, you may want to learn more about the process of fitting lasso models and how you can make modifications to this process.

[LASSO] **lasso fitting** The process (in a nutshell) of fitting lasso models

Acknowledgments

We thank Christian B. Hansen of the University of Chicago for his advice over the years and his amazingly fast responses to our numerous questions.

We thank Otilia Boldea of Tilburg University for her advice and for sharing her course materials on model-selection methods and big data.

We thank Damian Kozbur of the University of Zurich for his advice about how to select the lasso penalty parameter.

We thank Denis Chetverikov of the University of California at Los Angeles for conversations about the CV lasso.

We thank Victor Chernozhukov of the Massachusetts Institute of Technology and Alexandre Belloni of Duke University for their responses to our questions.

We thank Achim Ahrens of ETH Zürich and Immigration Policy Lab, Christian B. Hansen of the University of Chicago, and Mark E. Schaffer of Heriot-Watt University for their Stata commands `lasso2`, `cvlasso`, `rlasso`, `pdslasso`, `ivlasso`, and `lassologit`, which are discussed in Ahrens, Hansen, and Schaffer (2018, 2020). We also thank them for conversations about their approach to these methods.

We thank Wilbur Townsend of Harvard Business School for his Stata command `elasticregress`.

We thank Adrian Mander of the SPC (Statistical Process Control) for his Stata command `lars`.

We thank Jordi Sunyer and Mar Alvarez-Pedrerol of the ISGlobal Center for Research in Environmental Epidemiology and Raquel Garcia-Esteban of the RTI Health Solutions for sharing the data they used in Sunyer et al. (2017).

References

Ahrens, A., C. B. Hansen, and M. E. Schaffer. 2018. pdslasso: Stata module for post-selection and post-regularization OLS or IV estimation and inference. Statistical Software Components S458459, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s458459.html.

———. 2020. lassopack: Model selection and prediction with regularized regression in Stata. *Stata Journal* 20: 176–235.

Ahrens, A., C. B. Hansen, M. E. Schaffer, and T. Wiemann. 2024. ddml: Double/debiased machine learning in Stata. *Stata Journal* 24: 3–45.

Belloni, A., V. Chernozhukov, and C. B. Hansen. 2014. High-dimensional methods and inference on structural and treatment effects. *Journal of Economic Perspectives* 28: 29–50. https://doi.org/10.1257/jep.28.2.29.

Berk, R., L. D. Brown, A. Buja, K. Zhang, and L. Zhao. 2013. Valid post-selection inference. *Annals of Statistics* 41: 802–837. https://doi.org/10.1214/12-AOS1077.

Bühlmann, P., and S. van de Geer. 2011. *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Berlin: Springer.

Cameron, A. C., and P. K. Trivedi. 2022. *Microeconometrics Using Stata*. 2nd ed. College Station, TX: Stata Press.

Dallakyan, A. 2022. graphiclasso: Graphical lasso for learning sparse inverse-covariance matrices. *Stata Journal* 22: 625–642.

Drukker, D. M., and D. Liu. 2019. An introduction to the lasso in Stata. *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2019/09/09/an-introduction-to-the-lasso-in-stata/.

Hastie, T. J., R. J. Tibshirani, and M. Wainwright. 2015. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Boca Raton, FL: CRC Press. https://doi.org/10.1201/b18401.

Huang, W., Y. Wang, and L. Zhou. 2024. Identify latent group structures in panel data: The classifylasso command. *Stata Journal* 24: 46–71.

Javanmard, A., and A. Montanari. 2014. Confidence intervals and hypothesis testing for high-dimensional regression. *Journal of Machine Learning Research* 15: 2869–2909.

Lee, J. D., D. L. Sun, Y. Sun, and J. E. Taylor. 2016. Exact post-selection inference, with application to the lasso. *Annals of Statistics* 44: 907–927. https://doi.org/10.1214/15-AOS1371.

Leeb, H., and B. M. Pötscher. 2005. Model selection and inference: Facts and fiction. *Econometric Theory* 21: 21–59. https://doi.org/10.1017/S0266466605050036.

———. 2006. Can one estimate the conditional distribution of post-model-selection estimators? *Annals of Statistics* 34: 2554–2591. https://doi.org/10.1214/009053606000000821.

———. 2008. Sparse estimators and the oracle property, or the return of Hodges' estimator. *Journal of Econometrics* 142: 201–211. https://doi.org/10.1016/j.jeconom.2007.05.017.

Schwarz, C. 2023. Estimating text regressions using txtreg_train. *Stata Journal* 23: 799–812.

Sunyer, J., E. Suades-González, R. García-Esteban, I. Rivas, J. Pujol, M. Alvarez-Pedrerol, J. Forns, X. Querol, and X. Basagaña. 2017. Traffic-related air pollution and attention in primary school children: Short-term association. *Epidemiology* 28: 181–189. https://doi.org/10.1097/EDE.0000000000000603.

Tibshirani, R. J. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society,* B ser., 58: 267–288. https://doi.org/10.1111/j.2517-6161.1996.tb02080.x.

van de Geer, S., P. Bühlmann, Y. Ritov, and R. Dezeure. 2014. On asymptotically optimal confidence regions and tests for high-dimensional models. *Annals of Statistics* 42: 1166–1202. https://doi.org/10.1214/14-AOS1221.

Zhang, C.-H., and S. S. Zhang. 2014. Confidence intervals for low dimensional parameters in high dimensional linear models. *Journal of the Royal Statistical Society,* B ser., 76: 217–242. https://doi.org/10.1111/rssb.12026.

Also see

[LASSO] **Lasso inference intro** — Introduction to inferential lasso models

[H2OML] **h2oml** — Introduction to commands for Stata integration with H2O machine learning

Lasso inference intro — Introduction to inferential lasso models

Description

Lasso selects covariates and estimates coefficients but does not provide the standard errors required for performing statistical inference. Stata provides three additional lasso-based methods for estimating the coefficients and standard errors for a subset of the covariates, and the results have the added advantage of being estimates of values from the true model that generated the data being analyzed.

The methods are double selection, partialing out, and cross-fit partialing out, which is also known as double machine learning. They can be applied to linear, logistic, and Poisson regression models. Partialing out and cross-fit partialing out can also be used with endogenous covariates and instrumental variables in linear models.

Remarks and examples

Remarks are presented under the following headings:

The problem

You want to know the true effects of `z1` and `z2` on `y`, by which we mean the effect in the true underlying model that generated the data being analyzed. We specify two variables, but you could specify one or a handful.

You do not know whether `z1` and `z2` belong in the model, although you have your suspicions. Your problem is to estimate the effects (coefficients) of `z1` and `z2` and obtain their standard errors.

At the same time, you do not know the other variables that appear in the model, but you do know that they are among variables from `x1` to `500`. You also know that only a small number of them appear. We will be more precise about the meaning of "small" later.

Possible solutions

If you had a sufficient number of observations in your data, you could fit a linear regression of `y` on `z1`, `z2`, and `x1` to `x500`:

```
. regress y z1 z2 x1-x500
```

The above is a solution because including extra explanatory variables does not cause bias, at least as long as the number of covariates is not too large. Including extra variables merely causes a loss of efficiency. Except there may be no merely about it. You may not have a sufficient number of observations to fit this regression and answer questions about `z1` and `z2` with any degree of certainty.

In that case, you could use your scientific judgment to select the covariates that need to appear in the model:

```
. regress y z1 z2 x3 x9 x203 x333 x478
```

The problem here is that you must be correct about the variables you included and excluded. And the insight that led you to choose them cannot come from the data. And the choice you made is not testable. And theory seldom provides sufficient guidance about variables or their functional form. In the age of big data, modern research is increasingly looking for data-dependent guidance and rules.

Here is yet another solution. Select and fit the model using `lasso`, and force the inclusion of `z1` and `z2` with parentheses:

```
. lasso linear y (z1 z2) x1-x500
```

Now `lasso` will select a model and obtain its coefficients. Problem is, the lasso procedure does not provide standard errors. You cannot perform statistical tests of significance of `z1` and `z2` or obtain confidence intervals for them. And there is a reason for that. `lasso` does not account for mistakes in selecting from the potential covariates `x1` to `x500`. Any mistakes it makes in selecting covariates that are also correlated with `z1` or `z2` would lead to bias in estimating the coefficients and standard errors of `z1` and `z2`.

Here is a solution. Refit the model that `lasso` selected using `regress`. We would not recommend this. Everyone agrees that it would be better to split your data into two samples, use the first to select the model, and use the second to refit it. You will now have the standard errors you need. But even this will not provide good estimates and standard errors if your interest is in the true model that generated the data. The problem is twofold. First, this process still does not account sufficiently for the sampling variability of the selection process for variables from `x1` to `x500`. Second, it does not account for the possibility of small coefficients in the true model. This second problem is more common and more detrimental than you might guess. See, for instance, Leeb and Pötscher (2005, 2006, 2008) and Belloni, Chernozhukov, and Hansen (2014a).

Solutions that focus on the true model

If your interest is inference about `z1` and `z2` in the true model that generated the data, the solution is to type

```
. dsregress y z1 z2, controls(x1-x500)
```

or

```
. poregress y z1 z2, controls(x1-x500)
```

or

```
. xporegress y z1 z2, controls(x1-x500)
```

These commands produce the double selection, partialing-out, and cross-fit partialing-out solutions, respectively, for the linear model, but commands also exist for logistic, Poisson, and instrumental-variables regression. These solutions all use multiple lassos and moment conditions that are robust to the model-selection mistakes that lasso makes; namely, that it does not select the covariates of the true model with probability 1. Of the three, the cross-fit partialing-out solution is best, but it can take a long time to run. The other two solutions are most certainly respectable. The cross-fit solution allows the true model to have more coefficients, and it allows the number of potential covariates, `x1-x500` in our examples, to be much larger. Technically, cross-fit has a less restrictive sparsity requirement.

All three of the methods have a sparsity requirement, and we have advice for you.

1. Let the commands fit the lassos using the default method, which is `plugin`.
2. If meeting the sparsity requirement concerns you, use cross-fit partialing out.

You may think of sparsity requirements as being unique to lasso, but they are not. Think about fitting an ordinary logistic regression model or any other estimator with only asymptotic properties. How many variables can be reliably included in the model if you have 100 observations? 500 observations? 1,000? 10,000? There is no answer to those questions except to say more with 1,000 than 500, more with 10,000 than 1,000, and so on.

The story is the same with the three inference methods. We can tell you more observations are better, and we can tell you more. Their requirements are stricter than those for logistic regression. The sparsity requirement for double selection and partialing out is that

$$\frac{s}{\sqrt{N}/\ln p} \qquad \text{is small}$$

where s is the number of covariates in the true model, N is the number of observations in the data, and p is the number of potential covariates. The sparsity requirement for cross-fit partialing out is the same, except that $\sqrt{N}$ is replaced by N. It is that

$$\frac{s}{N/\ln p} \qquad \text{is small}$$

N is much larger than $\sqrt{N}$. That is why we said that, if meeting the sparsity requirement concerns you, use cross-fit partialing out. It allows more covariates for all values of N.

We recommended that the lassos be fit using plugins because plugins were developed with these three methods in mind. Plugins tend to produce models with fewer "extra" covariates. Fitting the lassos using cross-validation for selection, on the other hand, tends to include lots of extra covariates. Using the Bayesian information criterion function for selection tends to include a number of covariates that falls between the numbers selected by the other two methods.

All three methods report the estimated coefficients for `z1` and `z2`, their standard errors, test statistics, and confidence intervals. Understanding how they work will be easier if we reduce the number of variables from two to one. Let's consider obtaining estimates for α in the model

$$y = d\alpha + \mathbf{x}\boldsymbol{\beta} + \epsilon$$

where d is the covariate of interest.

The double-selection solution

Double selection is the easiest of the three to explain. Its algorithm is the following:

1. Run a lasso of d on $\mathbf{x}$.
2. Run a lasso of y on $\mathbf{x}$.
3. Let $\tilde{\mathbf{x}}$ be the union of the selected covariates from steps 1 and 2.
4. Regress y on d and $\tilde{\mathbf{x}}$.

The estimate of α and its test statistics are then the coefficient on d and its test statistics.

Step 1 is the extra selection step from which double selection gets its name. It is this step that causes the method to be robust to the mistakes in model selection that lasso makes.

Stata provides three double-selection commands—`dsregress`, `dslogit`, and `dspoisson`.

The partialing-out solution

The algorithm is the following:

1. Run a lasso of d on $\mathbf{x}$. Let $\tilde{\mathbf{x}}_d$ be the covariates selected.
2. Regress d on $\tilde{\mathbf{x}}_d$. Let $\tilde{d}$ be the residuals from this regression.
3. Run a lasso of y on $\mathbf{x}$. Let $\tilde{\mathbf{x}}_y$ be the covariates selected.
4. Regress y on $\tilde{\mathbf{x}}_y$. Let $\tilde{y}$ be the residuals from this regression.
5. Regress $\tilde{y}$ on $\tilde{d}$.

The estimate of α and its test statistics are then the coefficient on $\tilde{d}$ and its test statistics.

This algorithm is a high-dimensional version of the classic partialing-out estimator, which you can learn about in Wooldridge (2020, chap. 3-2). In the classic estimator, the moment conditions used to estimate the coefficient on d are orthogonal to the variables in $\mathbf{x}$. In the high-dimensional variant, the moment conditions in step 5 are orthogonal to the relevant variables in $\mathbf{x}$; thus, small changes in the variables included do not have a significant effect on the estimator for α.

Stata provides four partialing-out commands—`poregress`, `pologit`, `popoisson`, and `poivregress`.

`poivregress` provides a variation on the algorithm shown above that handles endogenous variables with instrumental variables in linear models.

The cross-fit partialing-out (double machine learning) solution

Cross-fit partialing out is a split-sample version of partialing out. Cross-fit partialing out is also known as double machine learning (DML).

1. Divide the data in roughly equal-sized subsamples 1 and 2.
2. In sample 1:
    a. Run a lasso of d on $\mathbf{x}$. Let $\tilde{\mathbf{x}}_{d1}$ be the covariates.
    b. Regress d on $\tilde{\mathbf{x}}_{d1}$. Let $\widehat{\boldsymbol{\beta}}_1$ be the estimated coefficients.
    c. Run a lasso of y on $\mathbf{x}$. Let $\tilde{\mathbf{x}}_{y1}$ be the covariates selected.
    d. Regress y on $\tilde{\mathbf{x}}_{y1}$. Let $\widehat{\gamma}_1$ be the estimated coefficients.
3. In sample 2:
    a. Fill in $\tilde{d} = d - \tilde{\mathbf{x}}_{d1}\widehat{\boldsymbol{\beta}}_1$.
    b. Fill in $\tilde{y} = y - \tilde{\mathbf{x}}_{y1}\hat{\gamma}_1$.

4. Still in sample 2:
   a. Run a lasso of d on $\mathbf{x}$. Let $\tilde{\mathbf{x}}_{d2}$ be the covariates.
   b. Regress d on $\tilde{\mathbf{x}}_{d2}$. Let $\widehat{\boldsymbol{\beta}}_2$ be the estimated coefficients.
   c. Run a lasso of y on $\mathbf{x}$. Let $\tilde{\mathbf{x}}_{y2}$ be the covariates selected.
   d. Regress y on $\tilde{\mathbf{x}}_{y2}$. Let $\widehat{\gamma}_2$ be the estimated coefficients.
5. In sample 1:
   a. Fill in $\tilde{d} = d - \tilde{\mathbf{x}}_{d2}\widehat{\boldsymbol{\beta}}_2$.
   b. Fill in $\tilde{y} = y - \tilde{\mathbf{x}}_{y2}\widehat{\gamma}_2$.
6. In the full sample: Regress $\tilde{y}$ on $\tilde{d}$.

The estimate of α and its test statistics are then the coefficient on $\tilde{d}$ and its test statistics.

Cross-fit partialing out has a more relaxed sparsity requirement than partialing out and double selection, as we mentioned earlier. This is because the sample is split and coefficients are obtained from one sample and used in another, which is independent, and that adds robustness.

There are two variants of cross-fit partialing out (recall it is also known as DML): DML1 and DML2. Shown above is the algorithm for DML2, which is Stata's default method. DML1, available as an option, predates DML2 and solves the moment conditions within each fold (group) that is cross-fit and then averages. DML2, by comparison, solves the moment conditions jointly. See *Methods and formulas*. DML2 produced better results in simulations in Chernozhukov et al. (2018).

In the algorithm shown, the sample is split in two. The software splits it into K parts, where $K = 10$ by default. You could specify $K = 2$, but you would not want to do that. Larger K works better and $K = 10$ is viewed as sufficient. This is known as the 10-fold method.

Stata provides four cross-fit partialing-out commands—`xporegress`, `xpologit`, `xpopoisson`, and `xpoivregress`. `xpoivregress` provides a variation on the algorithms that handles endogenous variables with instrumental variables in linear models.

Where to learn more

See

[LASSO] **dsregress**	Double-selection lasso linear regression
[LASSO] **dslogit**	Double-selection lasso logistic regression
[LASSO] **dspoisson**	Double-selection lasso Poisson regression
[LASSO] **poregress**	Partialing-out lasso linear regression
[LASSO] **pologit**	Partialing-out lasso logistic regression
[LASSO] **popoisson**	Partialing-out lasso Poisson regression
[LASSO] **poivregress**	Partialing-out lasso instrumental-variables regression
[LASSO] **xporegress**	Cross-fit partialing-out lasso linear regression
[LASSO] **xpologit**	Cross-fit partialing-out lasso logistic regression
[LASSO] **xpopoisson**	Cross-fit partialing-out lasso Poisson regression
[LASSO] **xpoivregress**	Cross-fit partialing-out lasso instrumental-variables regression

And then there is the literature.

For a strikingly readable introduction, see Belloni, Chernozhukov, and Hansen (2014a). For a more technical discussion, see Belloni and Chernozhukov (2011).

Double selection was developed by Belloni, Chernozhukov, and Hansen (2014b). Their article also provides first-rate intuition on why the process works.

Partialing out was developed by Belloni et al. (2012). The 2012 date makes it appear that partialing out predates double selection, but the ordering was due to different publication lags. Their article also develops the plugin estimator for lasso and then develops the partialing-out instrumental-variables estimator. Partialing out was extended from linear to nonlinear models by Belloni, Chernozhukov, and Wei (2016). For a three-page introduction to the partialing-out instrumental-variables estimator, see Chernozhukov, Hansen, and Spindler (2015).

Cross-fit partialing out was developed by Chernozhukov et al. (2018). The researchers of this article had worked on these issues for years. They later came together to assemble this important article, which is an odd but appealing mix of intuition and technical derivations, especially concerning sample splitting.

Bickel, Ritov, and Tsybakov (2009) predates all the above and provided the theoretical foundations for what would become the plugin estimator. It also provided rates of convergence for the lasso which was used by subsequent authors. The article is both seminal and technical.

References

Belloni, A., D. Chen, V. Chernozhukov, and C. B. Hansen. 2012. Sparse models and methods for optimal instruments with an application to eminent domain. *Econometrica* 80: 2369–2429. https://doi.org/10.3982/ECTA9626.

Belloni, A., and V. Chernozhukov. 2011. “High dimensional sparse econometric models: An Introduction”. In *Inverse Problems of High-Dimensional Estimation*, edited by P. Alguier, E. Gautier, and G. Stoltz, 121–156. Berlin: Springer. https://doi.org/10.1007/978-3-642-19989-9_3.

Belloni, A., V. Chernozhukov, and C. B. Hansen. 2014a. High-dimensional methods and inference on structural and treatment effects. *Journal of Economic Perspectives* 28: 29–50. https://doi.org/10.1257/jep.28.2.29.

———. 2014b. Inference on treatment effects after selection among high-dimensional controls. *Review of Economic Studies* 81: 608–650. https://doi.org/10.1093/restud/rdt044.

Belloni, A., V. Chernozhukov, and Y. Wei. 2016. Post-selection inference for generalized linear models with many controls. *Journal of Business and Economic Statistics* 34: 606–619. https://doi.org/10.1080/07350015.2016.1166116.

Bickel, P. J., Y. Ritov, and A. B. Tsybakov. 2009. Simultaneous analysis of Lasso and Dantzig selector. *Annals of Statistics* 37: 1705–1732. https://doi.org/10.1214/08-AOS620.

Chernozhukov, V., D. Chetverikov, M. Demirer, E. Duflo, C. B. Hansen, W. K. Newey, and J. M. Robins. 2018. Double/debiased machine learning for treatment and structural parameters. *Econometrics Journal* 21: C1–C68. https://doi.org/10.1111/ectj.12097.

Chernozhukov, V., C. B. Hansen, and M. Spindler. 2015. Post-selection and post-regularization inference in linear models with many controls and instruments. *American Economic Review* 105: 486–490. https://doi.org/10.1257/aer.p20151022.

Drukker, D. M., and D. Liu. 2019. Using the lasso for inference in high-dimensional models. *The Stata Blog: Not Elsewhere Classified.* https://blog.stata.com/2019/09/09/using-the-lasso-for-inference-in-high-dimensional-models/.

Leeb, H., and B. M. Pötscher. 2005. Model selection and inference: Facts and fiction. *Econometric Theory* 21: 21–59. https://doi.org/10.1017/S0266466605050036.

———. 2006. Can one estimate the conditional distribution of post-model-selection estimators? *Annals of Statistics* 34: 2554–2591. https://doi.org/10.1214/009053606000000821.

———. 2008. Sparse estimators and the oracle property, or the return of Hodges’ estimator. *Journal of Econometrics* 142: 201–211. https://doi.org/10.1016/j.jeconom.2007.05.017.

Wooldridge, J. M. 2020. *Introductory Econometrics: A Modern Approach*. 7th ed. Boston: Cengage.

Also see

[LASSO] **Lasso intro** — Introduction to lasso

bicplot — Plot Bayesian information criterion function after lasso

Description

`bicplot` graphs the Bayesian information criterion (BIC) function after a lasso fit.

`bicplot` can be used after `lasso`, `elasticnet`, `sqrtlasso`, `telasso`, or any of the lasso inference commands.

Quick start

Graph the BIC function after `lasso`, `sqrtlasso`, or `elasticnet`

```
bicplot
```

Graph the BIC function after `elasticnet` for $\alpha = 0.5$

```
bicplot, alpha(.5)
```

After any of the `ds` or `po` commands, graph the BIC function for the dependent variable `y`

```
bicplot, for(y)
```

After an `xpo` command without option `resample`, graph the BIC function for `x` in cross-fit fold 2

```
bicplot, for(x) xfold(2)
```

After an `xpo` command with `resample`, graph the BIC function for `x` in cross-fit fold 2 for the first resample

```
bicplot, for(x) xfold(2) resample(1)
```

Same as above, but graph the BIC function as a function of the ℓ_1-norm of the standardized coefficient vector

```
bicplot, for(x) xfold(2) resample(1) xunits(l1norm)
```

After `telasso`, graph the BIC function for the outcome variable `y` at treatment level 1

```
bicplot, for(y) tlevel(1)
```

Menu

Statistics > Postestimation

Syntax

After `lasso`, `sqrtlasso`, *and* `elasticnet`

`bicplot` [, *options*]

After `ds` *and* `po` *commands*

`bicplot, for(`*varspec*`)` [*options*]

After `xpo` *commands without* `resample`

`bicplot, for(`*varspec*`) xfold(#)` [*options*]

After `xpo` *commands with* `resample`

`bicplot, for(`*varspec*`) xfold(#) resample(#)` [*options*]

After `telasso` *for the outcome variable*

`bicplot, for(`*varspec*`) tlevel(#)` [*options*]

After `telasso` *for the treatment variable*

`bicplot, for(`*varspec*`)` [*options*]

After `telasso` *for the outcome variable with cross-fitting but without* `resample`

`bicplot, for(`*varspec*`) tlevel(#) xfold(#)` [*options*]

After `telasso` *for the treatment variable with cross-fitting but without* `resample`

`bicplot, for(`*varspec*`) xfold(#)` [*options*]

After `telasso` *for the outcome variable with cross-fitting and* `resample`

`bicplot, for(`*varspec*`) tlevel(#) xfold(#) resample(#)` [*options*]

After `telasso` *for the treatment variable with cross-fitting and* `resample`

`bicplot, for(`*varspec*`) xfold(#) resample(#)` [*options*]

varspec is *varname*, except after `poivregress` and `xpoivregress`, when it is either *varname* or `pred(`*varname*`)`.

options	Description
Main	
`xunits(`*x_unit_spec*`)`	x-axis units (scale); default is `xunits(rlnlambda)`, where `rlnlambda` denotes λ on a reverse logarithmic scale
`minmax`	add labels for the minimum and maximum x-axis units
* `for(`*varspec*`)`	lasso for *varspec*; `telasso`, `ds`, `po`, and `xpo` commands only
* `xfold(#)`	lasso for the #th cross-fit fold; `xpo` commands and `telasso` with `xfolds` only
* `resample(#)`	lasso for the #th resample; `xpo` commands and `telasso` with `resample` only
* `tlevel(#)`	lasso for the outcome model with the treatment level #; `telasso` only
`alpha(#)`	graph BIC function for $\alpha = \#$; default is the selected value α^*; allowed after `elasticnet` only
`lineopts(`*cline_options*`)`	affect rendition of the plotted lines
Reference lines	
`biclineopts(`*xline_options*`)`	affect rendition of reference line identifying the minimum of the BIC function or other stopping rule
`nobicline`	suppress reference line identifying the minimum of the BIC function or other stopping rule
`lslineopts(`*xline_options*`)`	affect rendition of reference line identifying the value selected using `lassoselect`
`nolsline`	suppress reference line identifying the value selected using `lassoselect`
`rlabelopts(`*r_label_opts*`)`	change look of labels for reference line
Data	
`data(`*filename* [`, replace`]`)`	save plot data to *filename*
Y axis, X axis, Titles, Legend, Overall	
twoway_options	any options other than `by()` documented in [G-3] ***twoway_options***

*`for(`*varspec*`)` is required for all `ds`, `po`, and `xpo` commands and for `telasso`.

`xfold(#)` is required for all `xpo` commands and for `telasso` when the option `xfolds(#)` was specified.

`resample(#)` is required for `xpo` and for `telasso` when the option `resample(#)` was specified.

`tlevel(#)` is required for the outcome model in `telasso`.

x_unit_spec	Description
`rlnlambda`	λ on a reverse logarithmic scale; the default
`lnlambda`	λ on a logarithmic scale
`l1norm`	ℓ_1-norm of standardized coefficient vector
`l1normraw`	ℓ_1-norm of unstandardized coefficient vector

xline_options	Description
style(*addedlinestyle*)	overall style of added line
[no]extend	[do not] extend line through plot region's margins
lstyle(*linestyle*)	overall style of line
lpattern(*linepatternstyle*)	line pattern (solid, dashed, etc.)
lwidth(*linewidthstyle*)	thickness of line
lcolor(*colorstyle*)	color and opacity of line

r_label_opts	Description
labgap(*size*)	margin between tick and label
labstyle(*textstyle*)	overall style of label
labsize(*textsizestyle*)	size of label
labcolor(*colorstyle*)	color and opacity of label

Options

Main

xunits(*x_unit_spec*) specifies the x-axis units used for graphing the BIC function. The following *x_unit_spec*s are available:

rlnlambda specifies x-axis units λ on a reverse logarithmic scale. This is the default.

lnlambda specifies x-axis units λ on a logarithmic scale.

l1norm specifies x-axis units ℓ_1-norm of the standardized coefficient vector.

l1normraw specifies x-axis units ℓ_1-norm of the unstandardized coefficient vector.

minmax adds labels for the minimum and maximum x-axis units to the graph of the BIC function.

for(*varspec*) specifies a particular lasso after telasso or a ds, po, or xpo estimation command fit using the option selection(bic). For all commands except poivregress and xpoivregress, *varspec* is always *varname*; it is either *depvar*, the dependent variable, or one of *varsofinterest* for which inference is done.

For poivregress and xpoivregress, *varspec* is either *varname* or pred(*varname*). The lasso for *depvar* is specified with its *varname*. For the endogenous variable *varname*, there are two lassos, which can be identified by *varname* and pred(*varname*). The exogenous variables of interest each have only one lasso, and it is specified by pred(*varname*).

This option is required after ds, po, and xpo commands.

xfold(#) specifies a particular lasso after an xpo estimation command. For each variable to be fit with a lasso, K lassos are done, one for each cross-fit fold, where K is the number of folds. This option specifies which fold, where $\# = 1, 2, \ldots, K$. xfold(#) is required after an xpo command.

resample(#) specifies a particular lasso after an xpo estimation command fit using the option resample(#). For each variable to be fit with a lasso, $R \times K$ lassos are done, where R is the number of resamples and K is the number of cross-fitting folds. This option specifies which resample, where $\# = 1, 2, \ldots, R$. resample(#), along with xfold(#), is required after an xpo command with resampling.

`tlevel(#)` specifies the lasso for the outcome variable at the specified treatment level after `telasso`. This option is required to refer to the outcome model after `telasso`.

`alpha(#)` graphs the BIC function for $\alpha = \#$. The default is `alpha(`α^*`)`, where α^* is the selected α. `alpha(#)` may only be specified after `elasticnet`.

`lineopts(`*cline_options*`)` affects the rendition of the plotted line. See [G-3] ***cline_options***.

Reference lines

`biclineopts(`*xline_options*`)` affects the rendition of the reference line identifying the minimum BIC value, the value selected when the stopping tolerance is reached, or the grid-minimum value.

xline_options are the following: `style(`*addedlinestyle*`)`, `noextend`, `lstyle(`*linestyle*`)`, `lpattern(`*linepatternstyle*`)`, `lwidth(`*linewidthstyle*`)`, and `lcolor(`*colorstyle*`)`. They specify how the reference line identifying the minimum BIC value is presented. See [G-4] ***addedlinestyle***, [G-4] ***linestyle***, [G-4] ***linepatternstyle***, [G-4] ***linewidthstyle***, and [G-4] ***colorstyle***.

`nobicline` suppresses the reference line identifying the minimum BIC value, the value selected when either the stopping tolerance or the grid-minimum value is reached.

`lslineopts(`*xline_options*`)` affects the rendition of the reference line identifying the value selected using `lassoselect`.

xline_options are the following: `style(`*addedlinestyle*`)`, `noextend`, `lstyle(`*linestyle*`)`, `lpattern(`*linepatternstyle*`)`, `lwidth(`*linewidthstyle*`)`, and `lcolor(`*colorstyle*`)`. They specify how the reference line identifying the value selected using `lassoselect` is presented. See [G-4] ***addedlinestyle***, [G-4] ***linestyle***, [G-4] ***linepatternstyle***, [G-4] ***linewidthstyle***, and [G-4] ***colorstyle***.

`nolsline` suppresses the reference line identifying the value selected using `lassoselect`.

`rlabelopts(`*r_label_opts*`)` changes the look of labels for the reference line. The label options `labgap(`*relativesize*`)`, `labstyle(`*textstyle*`)`, `labsize(`*textsizestyle*`)`, and `labcolor(`*colorstyle*`)` specify details about how the labels are presented. See [G-4] ***size***, [G-4] ***textstyle***, [G-4] ***textsizestyle***, and [G-4] ***colorstyle***.

Data

`data(`*filename* [`, replace`]`)` saves the plot data to a Stata data file.

Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G-3] ***twoway_options***, excluding `by()`. These include options for titling the graph (see [G-3] ***title_options***) and options for saving the graph to disk (see [G-3] ***saving_option***).

Remarks and examples

BIC plots graph the BIC function over the search grid for the lasso penalty parameter λ.

The search grid can be shown as the log of the lasso penalty parameter λ, `xunits(lnlambda)`; the reverse of that scale, `xunits(rlnlambda)`; the ℓ_1-norm of the standardized coefficients, `xunits(l1norm)`; or the ℓ_1-norm of the unstandardized coefficients, `xunits(l1normraw)`. The reverse log of lambda is the default because it represents the BIC search path over λ, with the first λ tried on the left and the last λ tried on the right.

BIC plots can be drawn after any command that directly searches over a grid of λ's. They can be drawn after the commands `lasso`, `elasticnet`, `sqrtlasso`, `telasso`, or any of the 11 lasso inference commands.

Examples that demonstrate how to use `bicplot` after the `lasso` command can be found in *BIC* in [LASSO] **lasso examples**.

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **lasso postestimation** — Postestimation tools for lasso for prediction

[CAUSAL] **telasso postestimation** — Postestimation tools for telasso

coefpath — Plot path of coefficients after lasso

Description

`coefpath` graphs the coefficient paths after any lasso fit using `selection(cv)`, `selection(adaptive)`, `selection(bic)`, or `selection(none)`. A line is drawn for each coefficient that traces its value over the searched values of the lasso penalty parameter λ or over the ℓ_1-norm of the fitted coefficients that result from lasso selection using those values of λ.

`coefpath` can be used after `lasso`, `elasticnet`, `sqrtlasso`, `telasso`, or any of the lasso inference commands.

Quick start

Graph the coefficient paths after `lasso`, `sqrtlasso`, or `elasticnet`

```
coefpath
```

Graph the unstandardized coefficient paths

```
coefpath, rawcoefs
```

Graph the coefficient paths after `elasticnet` for the $\alpha = 0.5$ lasso

```
coefpath, alpha(.5)
```

Same as above, but graph the paths using a single linestyle, rather than line-specific linestyles

```
coefpath, alpha(.5) mono
```

After any of the `ds` or `po` commands, graph the paths for the dependent variable `y`

```
coefpath, for(y)
```

Same as above, but graph the paths as a function of $\ln\lambda$

```
coefpath, for(y) xunits(lnlambda)
```

After an `xpo` command without `resample`, graph the paths for `x` in cross-fit fold 2

```
coefpath, for(x) xfold(2)
```

After an `xpo` command with `resample`, graph the paths for `x` in cross-fit fold 2 for the first resample

```
coefpath, for(x) xfold(2) resample(1)
```

After `telasso`, graph the paths for the outcome variable `y` at treatment level 1

```
coefpath, for(y) tlevel(1)
```

Menu

Statistics > Postestimation

Syntax

After `lasso`, `sqrtlasso`, *and* `elasticnet`

`coefpath` [, *options*]

After `ds` *and* `po` *commands*

`coefpath, for(`*varspec*`)` [*options*]

After `xpo` *commands without* `resample`

`coefpath, for(`*varspec*`) xfold(#)` [*options*]

After `xpo` *commands with* `resample`

`coefpath, for(`*varspec*`) xfold(#) resample(#)` [*options*]

After `telasso` *for the outcome variable*

`coefpath, for(`*varspec*`) tlevel(#)` [*options*]

After `telasso` *for the treatment variable*

`coefpath, for(`*varspec*`)` [*options*]

After `telasso` *for the outcome variable with cross-fitting but without* `resample`

`coefpath, for(`*varspec*`) tlevel(#) xfold(#)` [*options*]

After `telasso` *for the treatment variable with cross-fitting but without* `resample`

`coefpath, for(`*varspec*`) xfold(#)` [*options*]

After `telasso` *for the outcome variable with cross-fitting and* `resample`

`coefpath, for(`*varspec*`) tlevel(#) xfold(#) resample(#)` [*options*]

After `telasso` *for the treatment variable with cross-fitting and* `resample`

`coefpath, for(`*varspec*`) xfold(#) resample(#)` [*options*]

varspec is *varname*, except after `poivregress` and `xpoivregress`, when it is either *varname* or `pred(`*varname*`)`.

options	Description
Main	
xunits(*x_unit_spec*)	x-axis units (scale); default is xunits(l1norm)
minmax	adds minimum and maximum values to the x axis
* for(*varspec*)	lasso for *varspec*; telasso, ds, po, and xpo commands only
* xfold(#)	lasso for the #th cross-fit fold; xpo commands and telasso with xfolds only
* resample(#)	lasso for the #th resample; xpo commands and telasso with resample only
* tlevel(#)	lasso for the outcome model with the treatment level #; telasso only
alpha(#)	graph coefficient paths for $\alpha = \#$; default is the selected value α^*; only allowed after elasticnet
rawcoefs	graph unstandardized coefficient paths
Reference line	
rlopts(*cline_options*)	affect rendition of reference line
norefline	suppress plotting reference line
Path	
lineopts(*cline_options*)	affect rendition of all coefficient paths; not allowed when there are 100 or more coefficients
line#opts(*cline_options*)	affect rendition of coefficient path #; not allowed when there are 100 or more coefficients
mono	graph coefficient paths using a single line; default is mono for 100 or more coefficients
monoopts(*cline_options*)	affect rendition of line used to graph coefficient paths when mono is specified
Data	
data(*filename* [, replace])	save plot data to *filename*
Y axis, X axis, Titles, Legend, Overall	
twoway_options	any options other than by() documented in [G-3] ***twoway_options***

*for(*varspec*) is required for all ds, po, and xpo commands and for telasso.
xfold(#) is required for all xpo commands and for telasso when the option xfolds(#) was specified.
resample(#) is required for xpo and for telasso when the option resample(#) was specified.
tlevel(#) is required for the outcome model in telasso.

x_unit_spec	Description
l1norm	ℓ_1-norm of standardized coefficient vector; the default
l1normraw	ℓ_1-norm of unstandardized coefficient vector
lnlambda	λ on a logarithmic scale
rlnlambda	λ on a reverse logarithmic scale

Options

Main

`xunits(`*x_unit_spec*`)` specifies the x-axis units used for graphing the coefficient paths. The following *x_unit_spec*s are available:

`l1norm` specifies x-axis units ℓ_1-norm of the standardized coefficient vector. This is the default.

`l1normraw` specifies x-axis units ℓ_1-norm of the unstandardized coefficient vector.

`lnlambda` specifies x-axis units λ on a logarithmic scale.

`rlnlambda` specifies x-axis units λ on a reverse logarithmic scale.

`minmax` adds minimum and maximum values to the x axis.

`for(`*varspec*`)` specifies a particular lasso after `telasso` or after a `ds`, `po`, or `xpo` estimation command fit using the option `selection(cv)`, `selection(adaptive)`, or `selection(bic)`. For all commands except `poivregress` and `xpoivregress`, *varspec* is always *varname*.

For the `ds`, `po`, and `xpo` commands except `poivregress` and `xpoivregress`, *varspec* is either *depvar*, the dependent variable, or one of *varsofinterest* for which inference is done.

For `poivregress` and `xpoivregress`, *varspec* is either *varname* or `pred(`*varname*`)`. The lasso for *depvar* is specified with its *varname*. Each of the endogenous variables have two lassos, specified by *varname* and `pred(`*varname*`)`. The exogenous variables of interest each have only one lasso, and it is specified by `pred(`*varname*`)`.

For `telasso`, *varspec* is either the outcome variable or the treatment variable.

This option is required after `telasso` and after the `ds`, `po`, and `xpo` commands.

`xfold(#)` specifies a particular lasso after an `xpo` estimation command or after `telasso` when the option `xfolds(#)` was specified. For each variable to be fit with a lasso, K lassos are done, one for each cross-fit fold, where K is the number of folds. This option specifies which fold, where $\# = 1, 2, \ldots, K$. `xfold(#)` is required after an `xpo` command and after `telasso` when the option `xfolds(#)` was specified.

`resample(#)` specifies a particular lasso after an `xpo` estimation command or after `telasso` fit using the option `resample(#)`. For each variable to be fit with a lasso, $R \times K$ lassos are done, where R is the number of resamples and K is the number of cross-fitting folds. This option specifies which resample, where $\# = 1, 2, \ldots, R$. `resample(#)`, along with `xfold(#)`, is required after an `xpo` command and after `telasso` with resampling.

`tlevel(#)` specifies the lasso for the outcome variable at the specified treatment level after `telasso`. This option is required to refer to the outcome model after `telasso`.

`alpha(#)` graphs coefficient paths for $\alpha = \#$. The default is `alpha(`α^*`)`, where α^* is the selected α. `alpha(#)` may only be specified after `elasticnet`.

`rawcoefs` specifies that unstandardized coefficient paths be graphed. By default, coefficients of standardized variables (mean 0 and standard deviation 1) are graphed.

Reference line

`rlopts(`*cline_options*`)` affects the rendition of the reference line. See [G-3] ***cline_options***.

`norefline` suppresses plotting the reference line.

Path

`lineopts(`*cline_options*`)` affects the rendition of all coefficient paths. See [G-3] ***cline_options***. `lineopts()` is not allowed when there are 100 or more coefficients.

`line#opts(`*cline_options*`)` affects the rendition of coefficient path #. See [G-3] ***cline_options***. `line#opts()` is not allowed when there are 100 or more coefficients.

`mono` graphs the coefficient paths using a single line. `mono` is the default when there are 100 or more coefficients in the lasso.

`monoopts(`*cline_options*`)` affects the rendition of the line used to graph the coefficient paths when `mono` is specified. See [G-3] ***cline_options***.

Data

`data(`*filename* [`, replace`]`)` saves the plot data to a Stata data file.

Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G-3] ***twoway_options***, excluding `by()`. These include options for titling the graph (see [G-3] ***title_options***) and options for saving the graph to disk (see [G-3] ***saving_option***).

Remarks and examples

Remarks are presented under the following headings:

Coefficient path plots

Coefficient path plots show the path of each coefficient over the search grid for the lasso penalty parameter λ. The grid can be shown as either the log of lambda, `xunits(lnlambda)`; the reverse of that scale, `xunits(rlnlambda)`; the ℓ_1-norm of the standardized coefficients, `xunits(l1norm)` (the default); or the ℓ_1-norm of the unstandardized coefficients. The ℓ_1-norm of the standardized coefficients is traditionally the default because it directly represents the lasso constraint in the standardized coefficient space—the maximum allowed sum of the absolute values of the coefficients subject to a value of lambda. λ and the ℓ_1-norm have an inverse monotonic relationship. λ is the lasso penalty. The ℓ_1-norm is its impact on the length of the coefficient vector.

Coefficient path plots can be drawn after any command that directly searches over a grid of λ's—that is, after any command that uses option `selection(cv)`, `selection(adaptive)`, or `selection(none)`. They can be drawn after commands `lasso`, `elasticnet`, `sqrtlasso`, or any of the 11 lasso inference commands.

An example

We used the auto dataset to demonstrate the lasso command in [LASSO] **lasso**.

```
. sysuse auto
(1978 automobile data)
```

While this dataset is an unlikely candidate for fitting with lasso, it is perfectly good for demonstrating both lasso fitting and coefpath.

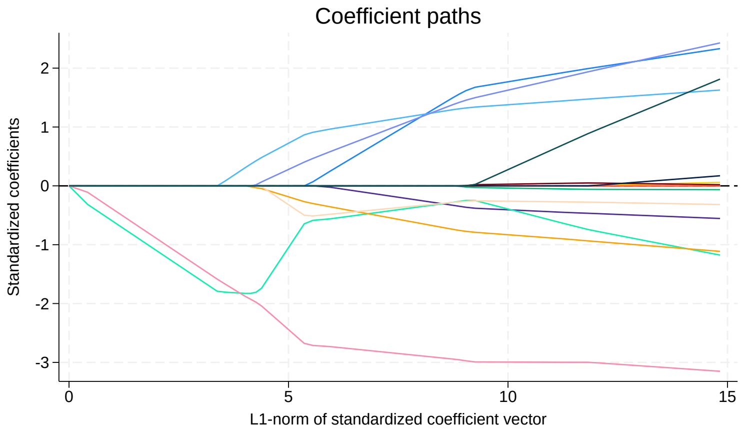
In that entry, we discussed how to model mpg on the remaining covariates in the dataset by typing

```
. lasso linear mpg i.foreign i.rep78 headroom weight turn gear_ratio price
> trunk length displacement, selection(cv, alllambdas) stop(0) rseed(12345)
Evaluating up to 100 lambdas in grid ...
Grid value 1:     lambda =  4.69114   no. of nonzero coef. =  0
  (output omitted)
Grid value 100:   lambda = .0004691   no. of nonzero coef. = 13
10-fold cross-validation with 100 lambdas ...
Fold  1 of 10:  10....20....30....40....50....60....70....80....90....100
  (output omitted)
Fold 10 of 10:  10....20....30....40....50....60....70....80....90....100
... cross-validation complete
Lasso linear model                           No. of obs         =         69
                                             No. of covariates =         15
Selection: Cross-validation                  No. of CV folds    =         10

─────────┬────────────────────────────────────────────────────────────────
         │                                No. of      Out-of-      CV mean
         │                               nonzero       sample   prediction
      ID │     Description      lambda     coef.    R-squared        error
─────────┼────────────────────────────────────────────────────────────────
       1 │    first lambda     4.69114         0       0.0049     33.74852
      40 │   lambda before    .1246008         8       0.6225     12.80314
    * 41 │ selected lambda    .1135316         8       0.6226     12.79854
      42 │    lambda after    .1034458         8       0.6218     12.82783
     100 │     last lambda    .0004691        13       0.5734     14.46932
─────────┴────────────────────────────────────────────────────────────────
* lambda selected by cross-validation.
```

This command is fully explained in [LASSO] **lasso**. Of special interest here is the suboption alllambdas and the option stop(0). Together, they ensure that the full 100 default values in the cross-validation grid are searched. Otherwise, lasso will stop searching once it has found an optimum or once one of its other stopping rules is met.

Graphing the coefficient paths for this lasso fit is as easy as typing

```
. coefpath
```

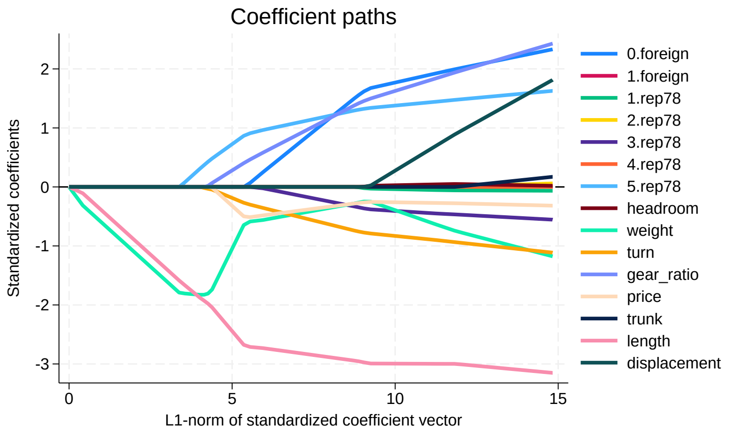


The x axis shows the sum of the absolute values of the penalized coefficients (the ℓ_1-norm) going from 0 to 15. Each line traces the penalized coefficient for one of the standardized covariates in our model. These graphs are popular but pose a bit of a conundrum. They can only be interpreted when there are few covariates, yet lasso is often most applicable when there are many covariates.

Adding a legend

Often, there are too many variables to allow for interest in any single path. These data are small enough that we can look at each covariate. Let's turn the legend on and place it beside the graph, using a single column for the keys,

```
. coefpath, lineopts(lwidth(thick)) legend(on)
```



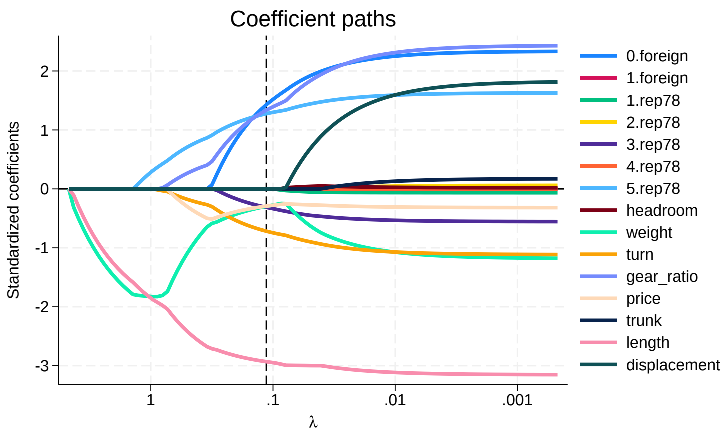
Looking at the graph, we now know which variable is traced by each line. We see that car weight is traced by the light green line that starts off downward before its effect declines toward 0. What is happening here is that weight enters early and absorbs any effect of other variables that are correlated with it but have yet to enter the model. When 5.rep78 enters the model, the coefficient on weight flattens. As gear_ratio, price, and turn enter, the effect of weight is further attenuated toward 0. This is simply what happens when correlated variables are added to a model. With lasso, they are added slowly because the lasso penalty brings in the coefficients in a penalized form rather than all at once.

Lasso is just letting variables into the model based on its penalty and the current value of lambda. We can see what is happening, but that is about it.

λ scale and reference line

In this example from [LASSO] **lasso**, we might find it yet more interesting to put our plot on the same scale as the cvplot from that entry and add a reference line for the λ selected by cross-validation. We change the scale by adding xunits(rlnlambda) and place the reference line by adding xline(.1135),

```
. coefpath, lineopts(lwidth(thick)) legend(on) xunits(rlnlambda) xline(.1135)
```



We know from the output of lasso that cross-validation selected eight coefficients. We can now see where each of them is in its path when cross-validation selected a model.

After fitting with sqrtlasso

There is not much to say about using coefpath after fitting with sqrtlasso. You type the same thing after sqrtlasso that you would type after lasso.

If you wish to see that, you can simply change lasso to sqrtlasso in the estimation command above. Make no changes to any other commands.

What's more, you can add the option sqrtlasso whenever it is allowed to any of the inference commands below. Nothing changes in the way we specify our coefpath commands.

After fitting with elasticnet

The only thing that changes with coefpath after an elasticnet command is that we can specify the option alpha() to graph the paths for a value of α that is different than the alpha chosen by elasticnet.

We can fit an `elasticnet` model using the auto dataset:

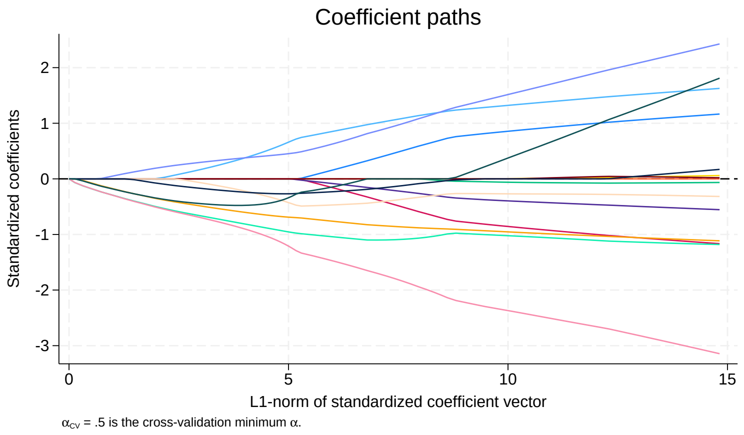
```
. elasticnet linear mpg i.foreign i.rep78 headroom weight turn gear_ratio
> price trunk length displacement,
> selection(cv, alllambdas) stop(0) rseed(12345)
```

(output omitted)

```
Elastic net linear model                         No. of obs        =         69
                                                 No. of covariates =         15
Selection: Cross-validation                      No. of CV folds   =         10

-------------------------------------------------------------------------------
               |                                 No. of    Out-of-      CV mean
               |                                nonzero     sample   prediction
alpha       ID |     Description      lambda      coef.  R-squared        error
---------------+---------------------------------------------------------------
1.000          |
             1 |    first lambda    9.382281          0    -0.0064     34.13399
           109 |     last lambda    .0004691         13     0.5734     14.46932
---------------+---------------------------------------------------------------
0.750          |
           110 |    first lambda    9.382281          0    -0.0064     34.13399
           218 |     last lambda    .0004691         14     0.5736     14.46276
---------------+---------------------------------------------------------------
0.500          |
           219 |    first lambda    9.382281          0    -0.0033     34.02853
           264 |   lambda before    .1647149         11     0.6328     12.45289
         * 265 | selected lambda    .1500821         11     0.6331     12.44435
           266 |    lambda after    .1367492         11     0.6331     12.44506
           327 |     last lambda    .0004691         14     0.5738      14.4564
-------------------------------------------------------------------------------
* alpha and lambda selected by cross-validation.
```

We see that cross-validation chose α to be 0.5. Had it chosen 1, the `elasticnet` would have reduced to `lasso`. To see the coefficient path graph for $\alpha = 0.5$, we simply type

```
. coefpath
```



That looks quite a bit different from the first graph we drew in this entry, which is the graph for lasso and would be the same as the graph we would get if we added the option `alpha(1)`.

If we wanted the graph for $\alpha = 0.75$, we would type

```
. coefpath, alpha(.75)
```

After fitting with inference commands

All postestimation tools, including `coefpath`, can be used after the `ds`, `ps`, and `xpo` inference commands. Of all the postestimation commands, `coefpath` is the least likely to be useful in this context. The inference commands use lassos to select control variables from a set of potential controls. Aside from diagnosing whether something pathological occurred in the lasso, you are not supposed to care which controls were selected, much less their coefficients, and even less the path of those coefficients. Regardless, you can draw coefficient path plots for any lasso run by an inference command.

We will use a few of the examples from [LASSO] **Inference examples** to show you what to type to create a coefficient path plot.

All these examples use `breathe.dta`, which attempts to measure the effect of nitrogen dioxide on the reaction time of school children. All these examples will run, but we dispense with the output here. If you are curious, run some.

To prepare the dataset, type

```
. use https://www.stata-press.com/data/r19/breathe
. do no2
```

All the `ds` (double-selection) and `po` (partialing-out) `coefpaths` are drawn in exactly the same way. To fit one of the double-selection models from [LASSO] **Inference examples**, we type

```
. dsregress react no2_class, controls($cc i.($fc)) selection(cv) rseed(12345)
```

Recall that we are using global macros `$cc` and `$fc` to hold our control variables. `$cc` holds the continuous controls, and `$fc` holds the factor-variable controls. Typing `$cc` simply substitutes the list of continuous controls into our command, and likewise for `$fc`. We write `i.($fc)` so that each of the variables in `$fc` is expanded into dummy variables for each distinct level of the variable.

To draw the coefficient path plot for the lasso of the dependent variable `react`, we type

```
. coefpath, for(react)
```

To draw the plot for the lasso of the variable of interest `no2_class`, we type

```
. coefpath, for(no2_class)
```

If we had fit the models via partialing out by typing `poregress` instead of `dsregress`, nothing would change. Typing `coefpath, for(react)` would still produce the coefficient path plot for the lasso of `react`, and typing `coefpath, for(no2_class)` would still produce the plot for `no2_class`.

What's more, what we type to plot coefficient paths does not change if our dependent variable were dichotomous and we had fit the model by using `dslogit` or `pologit`. Nor does it change if the dependent variable is a count and we fit the model by using `dspoisson` or `popoisson`.

Things do change if we fit the model by using the `xpo` (cross-fit partialing-out) estimators. The `xpo` estimators perform lots of lassos. Let's refit our original model using `xporegress`.

```
. xporegress react no2_class, controls($cc i.($fc)) selection(cv) rseed(12345)
  (output omitted)
```

To see the lassos that `xporegress` ran, we can use `lassoinfo`:

```
. lassoinfo, each

    Estimate: active
     Command: xporegress

---------------------------------------------------------------------------------
             |                                                           No. of
   Dependent |             Selection  xfold  Selection                 selected
    variable |      Model     method    no.  criterion       lambda   variables
-------------+-------------------------------------------------------------------
   no2_class |     linear         cv      1    CV min.     .1801304          14
   no2_class |     linear         cv      2    CV min.     .2561599          10
   no2_class |     linear         cv      3    CV min.     .2181624          13
   no2_class |     linear         cv      4    CV min.     .1963854          13
   no2_class |     linear         cv      5    CV min.     .2352711          11
   no2_class |     linear         cv      6    CV min.     .2663564          12
   no2_class |     linear         cv      7    CV min.     .1293717          16
   no2_class |     linear         cv      8    CV min.     .1722497          15
   no2_class |     linear         cv      9    CV min.      .264197           9
   no2_class |     linear         cv     10    CV min.     .1184878          16
       react |     linear         cv      1    CV min.     2.130811          19
       react |     linear         cv      2    CV min.     2.443412          16
       react |     linear         cv      3    CV min.     2.062956          17
       react |     linear         cv      4    CV min.     4.220311          13
       react |     linear         cv      5    CV min.     7.434224           8
       react |     linear         cv      6    CV min.     3.356193          14
       react |     linear         cv      7    CV min.     7.954354           6
       react |     linear         cv      8    CV min.     6.422852           8
       react |     linear         cv      9    CV min.     2.982171          15
       react |     linear         cv     10    CV min.     2.738883          18
---------------------------------------------------------------------------------
```

That's 20 lassos! `react` has 10 and `no2_class` has 10. There is one lasso for each variable for each cross-validation fold. The cross-validation folds are enumerated in the column titled `xfold no.`. To see the cross-validation plot for the third cross-validation fold for the variable `react`, we type

```
    . coefpath, for(react) xfold(3)
```

Change `react` to `no2_class` to see the plot for `no2_class`.

Feel free to plot all 18 other pairings of each variable with the cross-validation folds.

Again, it would not matter if we had fit `xpologit` or `xpopoisson` models. We type the same thing to see our coefficient path plots.

The cross-fit models can create even more lassos. We are willing to resample the whole process to reduce the sampling variability. Let's resample the process 10 times:

```
    . xporegress react no2_class, controls($cc i.($fc)) selection(cv)   ///
         resample(10) rseed(12345)
```

If you type that command, be patient; it takes a few minutes to run.

Now, let's look at our lassos:

```
    . lassoinfo, each

        Estimate: active
         Command: xporegress

    ─────────────────────────────────────────────────────────────────────────────
                │                                                         No. of
      Dependent │             Selection Resample  xfold Selection            sel.
       variable │      Model     method   number    no. criterion    lambda   var.
    ────────────┼────────────────────────────────────────────────────────────────
                │
      no2_class │     linear         cv        1      1   CV min.  .1801304     14
      no2_class │     linear         cv        1      2   CV min.  .2561599     10
     (output omitted)
      no2_class │     linear         cv        1     10   CV min.  .1184878     16
      no2_class │     linear         cv        2      1   CV min.  .2118238     12
     (output omitted)
      no2_class │     linear         cv        2     10   CV min.  .1773874     13
     (output omitted)
      no2_class │     linear         cv        3     10   CV min.  .1676957     13
          react │     linear         cv        1      1   CV min.  2.130811     19
     (output omitted)
          react │     linear         cv        1     10   CV min.  2.738883     18
          react │     linear         cv        2      1   CV min.  4.379673     14
     (output omitted)
          react │     linear         cv        2     10   CV min.  3.747121     14
          react │     linear         cv        3      1   CV min.  5.821677     11
     (output omitted)
          react │     linear         cv        3     10   CV min.  3.668243     13
    ─────────────────────────────────────────────────────────────────────────────
```

We now have 30 of them! There is one for each variable within each cross-validation sample within each resample sample. Here is how we would graph the coefficient path plot for the third cross-validation sample in the second resample sample for the covariate of interest `no2_class`.

```
    . coefpath, for(no2_class) resample(2) xfold(3)
```

If we had typed `resample(10)` instead of `resample(3)` on our `xporegress` command, we would have 200 possible graphs. Have fun looking at those.

Yet again, it would not matter if we had fit `xpologit` or `xpopoisson` models. We still type the same thing to see our coefficient path plots.

Also see

[LASSO] **lasso postestimation** — Postestimation tools for lasso for prediction

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[CAUSAL] **telasso postestimation** — Postestimation tools for telasso

Collinear covariates — Treatment of collinear covariates

Description

Lasso, square-root lasso, and elastic net treat collinear covariates differently from traditional estimators. With these models, you specify variables that might be included in the model, and they choose the variables to be included. When you specify those variables, it is important that you present them with all possible alternatives. This means that, when including factor variables, you must include the full collinear set of indicators.

If you use Stata's factor-variable notation, it is handled automatically for you. If you create indicator variables for yourself, you must create and include them all.

Remarks and examples

Remarks are presented under the following headings:

Summary

Consider factor variable group that takes on the values 1, 2, and 3. If you type

```
. lasso linear y i.group ...
```

lasso will know that separate covariates for group 1, 2, and 3 are to be included among the variables to be potentially included in the model.

If you create your own indicator variables, you need to create and specify indicators for all the values of the factor variable:

```
. generate g1 = (group==1)
. generate g2 = (group==2)
. generate g3 = (group==3)
. lasso linear y g1 g2 g3 ...
```

It is important that you do not omit one of them, say, g1, and instead type

```
. lasso linear y g2 g3 ...
```

Explanation

With no loss of generality, we will focus on lasso for this explanation. Assume lasso has just found the best model for λ_i with k covariates and is now searching for the best model for λ_{i+1}, where $\lambda_{i+1} < \lambda_i$.

The λ_{i+1} model will not always be the same λ_i model with new covariates added, but this is often the case. (Sometimes, covariates in the λ_i model are removed.) Assume this is a case of adding only new covariates. Also assume that g1, g2, and g3 have not been chosen yet and that lasso chooses g1.

But what if we did not specify g1 among the potential covariates? What if rather than typing

```
. lasso linear y g1 g2 g3 ...
```

we typed

```
. lasso linear y g2 g3 ...
```

In that case, lasso would not choose g1 because it could not. It would choose some other covariate or covariates, perhaps g2, perhaps g3, perhaps g2 and g3, or perhaps other covariates. And lasso is on an inferior path because g1 was not among the potential covariates.

Although selecting both g2 and g3 in place of g1 gives an equivalent model for prediction, it may have wasted an extra penalty on the coefficients for g2 and g3. A model with only g1 may have a smaller penalty and allow other covariates to be included, which a model with g2 and g3 would not. By eliminating g1, we have denied lasso the opportunity to find a more parsimonious model.

Applies to inferential commands

You must also specify full collinear sets of potential covariates with the inferential commands. Specify full sets in the `controls()` option, such as

```
. dsregress y z1 z2, controls(g1 g2 g3 ...)
```

Likewise for the high-dimensional instruments in `poivregress` and `xpoivregress`:

```
. poivregress y ... (z1 z2 = g1 g2 g3 ...), controls(...)
```

Just as with `lasso`, the issue is handled automatically if you use factor-variable notation:

```
. dsregress y z1 z2, controls(i.group ...)
```

Does not apply to alwaysvars

With any lasso, you can specify covariates that will always appear in the model. You specify them in parentheses. For example, for `lasso`, type

```
. lasso linear y (x1 x2) x3 x4 ...
```

and for the inference commands, type

```
. dsregress y z1 z2, controls((x1 x2) x3 x4 ...)
```

We call the covariates that always appear in the model *alwaysvars*. The *alwaysvars* do not need to be full collinear sets. Indeed, collinear variables among the *alwaysvars* will be omitted.

Factor-variable notation handles the problem automatically in both cases:

```
. lasso linear (i.region ...) i.group ...
```

A base level will be set for `i.region` (or you can set it explicitly). For `i.group`, all levels will be included. If you try to set a base level for `i.group`, it will be ignored.

Also see

[LASSO] **lasso** — Lasso for prediction and model selection

[LASSO] **lasso examples** — Examples of lasso for prediction

cvplot — Plot cross-validation function after lasso

Description

`cvplot` graphs the cross-validation (CV) function after a lasso fit using `selection(cv)`, `selection(adaptive)`, `selection(bic)`, or `selection(none)`.

`cvplot` can be used after `lasso`, `elasticnet`, `sqrtlasso`, `telasso`, or any of the lasso inference commands.

Quick start

Graph the CV function after `lasso`, `sqrtlasso`, or `elasticnet`

```
cvplot
```

Same as above, and draw a reference line identifying the value selected by the one-standard-error rule

```
cvplot, seline
```

Graph the CV function after `elasticnet` for the $\alpha = 0.5$ lasso

```
cvplot, alpha(.5)
```

After any of the `ds` or `po` commands, graph the CV function for the dependent variable `y`

```
cvplot, for(y)
```

Same as above, and show standard error bands for the CV function

```
cvplot, for(y) se
```

After an `xpo` command without `resample`, graph the CV function for `x` in cross-fit fold 2

```
cvplot, for(x) xfold(2)
```

After an `xpo` command with `resample`, graph the CV function for `x` in cross-fit fold 2 for the first resample

```
cvplot, for(x) xfold(2) resample(1)
```

Same as above, but graph the CV function as a function of the ℓ_1-norm of the standardized coefficient vector

```
cvplot, for(x) xfold(2) resample(1) xunits(l1norm)
```

After `telasso`, graph the CV function for the outcome variable `x` at treatment level 1

```
cvplot, for(y) tlevel(1)
```

Menu

Statistics > Postestimation

Syntax

After `lasso`, `sqrtlasso`, *and* `elasticnet`

`cvplot` [, *options*]

After `ds` *and* `po` *commands*

`cvplot, for(`*varspec*`)` [*options*]

After `xpo` *commands without* `resample`

`cvplot, for(`*varspec*`) xfold(#)` [*options*]

After `xpo` *commands with* `resample`

`cvplot, for(`*varspec*`) xfold(#) resample(#)` [*options*]

After `telasso` *for the outcome variable*

`cvplot, for(`*varspec*`) tlevel(#)` [*options*]

After `telasso` *for the treatment variable*

`cvplot, for(`*varspec*`)` [*options*]

After `telasso` *for the outcome variable with cross-fitting but without* `resample`

`cvplot, for(`*varspec*`) tlevel(#) xfold(#)` [*options*]

After `telasso` *for the treatment variable with cross-fitting but without* `resample`

`cvplot, for(`*varspec*`) xfold(#)` [*options*]

After `telasso` *for the outcome variable with cross-fitting and* `resample`

`cvplot, for(`*varspec*`) tlevel(#) xfold(#) resample(#)` [*options*]

After `telasso` *for the treatment variable with cross-fitting and* `resample`

`cvplot, for(`*varspec*`) xfold(#) resample(#)` [*options*]

varspec is *varname*, except after `poivregress` and `xpoivregress`, when it is either *varname* or `pred(`*varname*`)`.

options	Description
Main	
xunits(*x_unit_spec*)	x-axis units (scale); default is xunits(rlnlambda), where rlnlambda denotes λ on a reverse logarithmic scale
minmax	add labels for the minimum and maximum x-axis units
* for(*varspec*)	lasso for *varspec*; telasso, ds, po, and xpo commands only
* xfold(#)	lasso for the #th cross-fit fold; xpo commands and telasso with xfolds only
* resample(#)	lasso for the #th resample; xpo commands and telasso with resample only
* tlevel(#)	lasso for the outcome model with the treatment level #; telasso only
alpha(#)	graph CV function for $\alpha = \#$; default is the selected value α^*; allowed after elasticnet only
lineopts(*cline_options*)	affect rendition of the plotted lines
S.E. plot	
se	show standard error bands for the CV function
seopts(*rcap_options*)	affect rendition of the standard error bands
Reference lines	
cvlineopts(*cline_options*)	affect rendition of reference line identifying the minimum of the CV function or other stopping rule
nocvline	suppress reference line identifying the minimum of the CV function or other stopping rule
lslineopts(*cline_options*)	affect rendition of reference line identifying the value selected using lassoselect
nolsline	suppress reference line identifying the value selected using lassoselect
selineopts(*cline_options*)	affect rendition of reference line identifying the value selected by the one-standard-error rule
[no]seline	draw or suppress reference line identifying the value selected by the one-standard-error rule; shown by default for selection(cv, serule)
hrefline	add horizontal reference lines that intersect the vertical reference lines
rlabelopts(*r_label_opts*)	change look of labels for reference line
Data	
data(*filename* [, replace])	save plot data to *filename*
Y axis, X axis, Titles, Legend, Overall	
twoway_options	any options other than by() documented in [G-3] ***twoway_options***

*for(*varspec*) is required for all ds, po, and xpo commands and for telasso.

xfold(#) is required for all xpo commands and for telasso when the option xfolds(#) was specified.

resample(#) is required for xpo and for telasso when the option resample(#) was specified.

tlevel(#) is required for the outcome model in telasso.

x_unit_spec	Description
`rlnlambda`	λ on a reverse logarithmic scale; the default
`lnlambda`	λ on a logarithmic scale
`l1norm`	ℓ_1-norm of standardized coefficient vector
`l1normraw`	ℓ_1-norm of unstandardized coefficient vector

r_label_opts	Description
`labgap(`*size*`)`	margin between tick and label
`labstyle(`*textstyle*`)`	overall style of label
`labsize(`*textsizestyle*`)`	size of label
`labcolor(`*colorstyle*`)`	color and opacity of label

Options

Main

`xunits(`*x_unit_spec*`)` specifies the x-axis units used for graphing the CV function. The following *x_unit_spec*s are available:

`rlnlambda` specifies x-axis units λ on a reverse logarithmic scale. This is the default.

`lnlambda` specifies x-axis units λ on a logarithmic scale.

`l1norm` specifies x-axis units ℓ_1-norm of the standardized coefficient vector.

`l1normraw` specifies x-axis units ℓ_1-norm of the unstandardized coefficient vector.

`minmax` adds labels for the minimum and maximum x-axis units to the graph of the CV function.

`for(`*varspec*`)` specifies a particular lasso after `telasso` or after a `ds`, `po`, or `xpo` estimation command fit using the option `selection(cv)`, `selection(adaptive)`, or `selection(bic)`. For all commands except `poivregress` and `xpoivregress`, *varspec* is always *varname*.

For the `ds`, `po`, and `xpo` commands except `poivregress` and `xpoivregress`, *varspec* is either *depvar*, the dependent variable, or one of *varsofinterest* for which inference is done.

For `poivregress` and `xpoivregress`, *varspec* is either *varname* or `pred(`*varname*`)`. The lasso for *depvar* is specified with its *varname*. Each of the endogenous variables have two lassos, specified by *varname* and `pred(`*varname*`)`. The exogenous variables of interest each have only one lasso, and it is specified by `pred(`*varname*`)`.

For `telasso`, *varspec* is either the outcome variable or the treatment variable.

This option is required after `telasso` and after the `ds`, `po`, and `xpo` commands.

`xfold(#)` specifies a particular lasso after an `xpo` estimation command or after `telasso` when the option `xfolds(#)` was specified. For each variable to be fit with a lasso, K lassos are done, one for each cross-fit fold, where K is the number of folds. This option specifies which fold, where $\# = 1, 2, \ldots, K$. `xfold(#)` is required after an `xpo` command and after `telasso` when the option `xfolds(#)` was specified.

`resample(#)` specifies a particular lasso after an `xpo` estimation command or after `telasso` fit using the option `resample(#)`. For each variable to be fit with a lasso, $R \times K$ lassos are done, where R is the number of resamples and K is the number of cross-fitting folds. This option specifies which resample, where $\# = 1, 2, \ldots, R$. `resample(#)`, along with `xfold(#)`, is required after an `xpo` command and after `telasso` with resampling.

`tlevel(#)` specifies the lasso for the outcome variable at the specified treatment level after `telasso`. This option is required to refer to the outcome model after `telasso`.

`alpha(#)` graphs the CV function for $\alpha = \#$. The default is `alpha(`α^*`)`, where α^* is the selected α. `alpha(#)` may only be specified after `elasticnet`.

`lineopts(`*cline_options*`)` affects the rendition of the plotted line. See [G-3] ***cline_options***.

S.E. plot

`se` shows standard error bands for the CV function.

`seopts(`*rcap_options*`)` affects the rendition of the standard error bands. See [G-3] ***rcap_options***.

Reference lines

`cvlineopts(`*cline_options*`)` affects the rendition of the reference line identifying the minimum CV value, the value selected when the stopping tolerance is reached, or the grid-minimum value. See [G-3] ***cline_options***.

`nocvline` suppresses the reference line identifying the minimum CV value, the value selected when the stopping tolerance is reached, or the grid-minimum value.

`lslineopts(`*cline_options*`)` affects the rendition of the reference line identifying the value selected using `lassoselect`. See [G-3] ***cline_options***.

`nolsline` suppresses the reference line identifying the value selected using `lassoselect`.

`selineopts(`*cline_options*`)` affects the rendition of the reference line identifying the value selected by the one-standard-error rule. See [G-3] ***cline_options***.

[`no`]`seline` draws or suppresses a reference line identifying the value selected by the one-standard-error rule. By default, the line is shown when `selection(cv, serule)` was the selection method for the lasso. For other selection methods, the line is not shown by default.

`hrefline` adds horizontal reference lines that intersect the vertical reference lines.

`rlabelopts(`*r_label_opts*`)` changes the look of labels for the reference line. The label options `labgap(`*relativesize*`)`, `labstyle(`*textstyle*`)`, `labsize(`*textsizestyle*`)`, and `labcolor(`*colorstyle*`)` specify details about how the labels are presented. See [G-4] ***size***, [G-4] ***textstyle***, [G-4] ***textsizestyle***, and [G-4] ***colorstyle***.

Data

`data(`*filename* [`, replace`]`)` saves the plot data to a Stata data file.

Y axis, X axis, Titles, Legend, Overall

twoway_options are any of the options documented in [G-3] ***twoway_options***, excluding `by()`. These include options for titling the graph (see [G-3] ***title_options***) and options for saving the graph to disk (see [G-3] ***saving_option***).

Remarks and examples

CV plots graph the CV function over the search grid for the lasso penalty parameter λ. For linear models, the CV function is the mean squared error of the predictions in the CV samples. For logit and Poisson models, the CV function is the mean deviance in the CV samples.

The search grid can be shown as the log of the lasso penalty parameter λ, `xunits(lnlambda)`; the reverse of that scale, `xunits(rlnlambda)`; the ℓ_1-norm of the standardized coefficients, `xunits(l1norm)`; or the ℓ_1-norm of the unstandardized coefficients, `xunits(l1normraw)`. The reverse log of lambda is the default because it represents the CV search path over λ, with the first λ tried on the left and the last λ tried on the right.

CV plots can be drawn after any command that directly searches over a grid of λ's—that is, after any command that used the option `selection(cv)`, `selection(adaptive)`, or `selection(none)`. They can be drawn after commands `lasso`, `elasticnet`, `sqrtlasso`, `telasso`, or any of the 11 lasso inference commands.

Examples that demonstrate how to use `cvplot` after the `lasso` command can be found in *The CV function* in [LASSO] **lasso**.

Examples after `elasticnet` can be found starting in example 2 of [LASSO] **elasticnet**.

Also see

[LASSO] **lasso postestimation** — Postestimation tools for lasso for prediction

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[CAUSAL] **telasso postestimation** — Postestimation tools for telasso

dslogit — Double-selection lasso logistic regression

Description	Quick start	Menu	Syntax	Options
Remarks and examples	Stored results	Methods and formulas	Reference	Also see

Description

`dslogit` fits a lasso logistic regression model and reports odds ratios along with standard errors, test statistics, and confidence intervals for specified covariates of interest. The double-selection method is used to estimate effects for these variables and to select from potential control variables to be included in the model.

Quick start

Report an odds ratio from a logistic regression of `y` on `d1`, and include `x1` to `x100` as potential control variables to be selected by lassos

```
dslogit y d1, controls(x1-x100)
```

Same as above, and estimate odds ratios for the levels of categorical `d2`

```
dslogit y d1 i.d2, controls(x1-x100)
```

Use cross-validation (CV) instead of a plugin iterative formula to select the optimal λ^* in each lasso

```
dslogit y d1 i.d2, controls(x1-x100) selection(cv)
```

Same as above, and set a random-number seed for reproducibility

```
dslogit y d1 i.d2, controls(x1-x100) selection(cv) rseed(28)
```

Specify CV for the lasso for `y` only, with the stopping rule criterion turned off

```
dslogit y d1 i.d2, controls(x1-x100) lasso(y, selection(cv), stop(0))
```

Same as above, but apply the option to the lassos for `y`, `d1`, and `i.d2`

```
dslogit y d1 i.d2, controls(x1-x100) lasso(*, selection(cv), stop(0))
```

Compute lassos beyond the CV minimum to get full coefficient paths, knots, etc.

```
dslogit y d1 i.d2, controls(x1-x100) lasso(*, selection(cv, alllambdas))
```

Menu

Statistics > Lasso > Lasso inferential models > Binary outcomes > Double-selection logit model

Syntax

dslogit *depvar varsofinterest* [*if*] [*in*] ,
  controls([(*alwaysvars*)] *othervars*) [*options*]

varsofinterest are variables for which coefficients and their standard errors are estimated.

options	Description
Model	
* controls([(*alwaysvars*)] *othervars*)	*alwaysvars* and *othervars* make up the set of control variables; *alwaysvars* are always included; lassos choose whether to include or exclude *othervars*
selection(plugin)	use a plugin iterative formula to select an optimal value of the lasso penalty parameter λ^* for each lasso; the default
selection(cv)	use CV to select an optimal value of the lasso penalty parameter λ^* for each lasso
selection(adaptive)	use adaptive lasso to select an optimal value of the lasso penalty parameter λ^* for each lasso
selection(bic)	use BIC to select an optimal value of the lasso penalty parameter λ^* for each lasso
sqrtlasso	use square-root lassos for *varsofinterest*
missingok	after fitting lassos, ignore missing values in any *othervars* not selected, and include these observations in the final model
offset(*varname*)	include *varname* in the lasso and model for *depvar* with its coefficient constrained to be 1
SE/Robust	
vce(*vcetype*)	*vcetype* may be robust (the default), cluster *clustvar*, or oim
Reporting	
level(#)	set confidence level; default is level(95)
or	report odds ratios; the default
coef	report estimated coefficients
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
Optimization	
[no]log	display or suppress an iteration log
verbose	display a verbose iteration log
rseed(#)	set random-number seed
Advanced	
lasso(*varlist*, *lasso_options*)	specify options for the lassos for variables in *varlist*; may be repeated
sqrtlasso(*varlist*, *lasso_options*)	specify options for square-root lassos for variables in *varlist*; may be repeated

`reestimate`	refit the model after using `lassoselect` to select a different λ^*
`noheader`	do not display the header on the coefficient table
`coeflegend`	display legend instead of statistics

*`controls()` is required.

varsofinterest, *alwaysvars*, and *othervars* may contain factor variables. Base levels of factor variables cannot be set for *alwaysvars* and *othervars*. See **[U] 11.4.3 Factor variables**.

`collect` is allowed; see **[U] 11.1.10 Prefix commands**.

`reestimate`, `noheader`, and `coeflegend` do not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

`controls(`[`(`*alwaysvars*`)`] *othervars*`)` specifies the set of control variables, which control for omitted variables. Control variables are also known as confounding variables. `dslogit` fits lassos for *depvar* and each of the *varsofinterest*. *alwaysvars* are variables that are always to be included in these lassos. *alwaysvars* are optional. *othervars* are variables that each lasso will choose to include or exclude. That is, each lasso will select a subset of *othervars*. The selected subset of *othervars* may differ across lassos. `controls()` is required.

`selection(plugin | cv | adaptive | bic)` specifies the selection method for choosing an optimal value of the lasso penalty parameter λ^* for each lasso or square-root lasso estimation. Separate lassos are estimated for *depvar* and each variable in *varsofinterest*. Specifying `selection()` changes the selection method for all of these lassos. You can specify different selection methods for different lassos using the option `lasso()` or `sqrtlasso()`. When `lasso()` or `sqrtlasso()` is used to specify a different selection method for the lassos of some variables, they override the global setting made using `selection()` for the specified variables.

`selection(plugin)` is the default. It selects λ^* based on a "plugin" iterative formula dependent on the data. See [LASSO] **lasso options**.

`selection(cv)` selects the λ^* that gives the minimum of the CV function. See [LASSO] **lasso options**.

`selection(adaptive)` selects λ^* using the adaptive lasso selection method. It cannot be specified when `sqrtlasso` is specified. See [LASSO] **lasso options**.

`selection(bic)` selects the λ^* that gives the minimum of the BIC function. See [LASSO] **lasso options**.

`sqrtlasso` specifies that square-root lassos be done rather than regular lassos for the *varsofinterest*. This option does not apply to *depvar*. Square-root lassos are linear models, and the lasso for *depvar* is always a logit lasso. The option `lasso()` can be used with `sqrtlasso` to specify that regular lasso be done for some variables, overriding the global `sqrtlasso` setting for these variables. See [LASSO] **lasso options**.

`missingok` specifies that, after fitting lassos, the estimation sample be redefined based on only the nonmissing observations of variables in the final model. In all cases, any observation with missing values for *depvar*, *varsofinterest*, *alwaysvars*, and *othervars* is omitted from the estimation sample for the lassos. By default, the same sample is used for calculation of the coefficients of the *varsofinterest* and their standard errors.

When `missingok` is specified, the initial estimation sample is the same as the default, but the sample used for the calculation of the coefficients of the *varsofinterest* can be larger. Now observations with missing values for any *othervars* not selected will be added to the estimation sample (provided there are no missing values for any of the variables in the final model).

`missingok` may produce more efficient estimates when data are missing completely at random. It does, however, have the consequence that estimation samples can change when selected variables differ in models fit using different selection methods. That is, when *othervars* contain missing values, the estimation sample for a model fit using the default `selection(plugin)` will likely differ from the estimation sample for a model fit using, for example, `selection(cv)`.

`offset(`*varname*`)` specifies that *varname* be included in the lasso and model for *depvar* with its coefficient constrained to be 1.

SE/Robust

`vce(`*vcetype*`)` specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (`robust`), that allow for intragroup correlation (`vce(cluster` *clustvar*`)`), and that are derived from asymptotic theory (`vce(oim)`). See [R] ***vce_option***.

When `vce(cluster` *clustvar*`)` is specified, all lassos also account for clustering. For each lasso, this affects how the log-likelihood function is computed and how the sample is split in cross-validation; see *Methods and formulas* in [LASSO] **lasso**. Specifying `vce(cluster` *clustvar*`)` may lead to different selected controls and therefore to different point estimates for your variable of interest when compared to the estimation that ignores clustering.

Reporting

`level(#)`; see [R] **Estimation options**.

`or` reports the estimated coefficients transformed to odds ratios, that is, e^{α}. Standard errors and confidence intervals are similarly transformed. `or` is the default.

`coef` reports the estimated coefficients α rather than the odds ratios (e^{α}). This option affects how results are displayed, not how they are estimated. `coef` may be specified at estimation or when replaying previously estimated results.

display_options: `noci`, `nopvalues`, `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `fvwrap(#)`, `fvwrapon(`*style*`)`, `cformat(%`*fmt*`)`, `pformat(%`*fmt*`)`, `sformat(%`*fmt*`)`, and `nolstretch`; see [R] **Estimation options**.

Optimization

[`no`]`log` displays or suppresses a log showing the progress of the estimation. By default, one-line messages indicating when each lasso estimation begins are shown. Specify `verbose` to see a more detailed log.

`verbose` displays a verbose log showing the iterations of each lasso estimation. This option is useful when doing `selection(cv)` or `selection(adaptive)`. It allows you to monitor the progress of the lasso estimations for these selection methods, which can be time consuming when there are many *othervars* specified in `controls()`.

`rseed(#)` sets the random-number seed. This option can be used to reproduce results for `selection(cv)` and `selection(adaptive)`. The default selection method `selection(plugin)` does not use random numbers. `rseed(#)` is equivalent to typing `set seed #` prior to running `dslogit`. See [R] **set seed**.

Advanced

`lasso(`*varlist*`,` *lasso_options*`)` lets you set different options for different lassos, or advanced options for all lassos. You specify a *varlist* followed by the options you want to apply to the lassos for these variables. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. `_all` or `*` may be used to specify *depvar* and all *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `lasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. It also overrides the global `sqrtlasso` option for these variables. See [LASSO] **lasso options**.

`sqrtlasso(`*varlist*`,` *lasso_options*`)` works like the option `lasso()`, except square-root lassos for the variables in *varlist* are done rather than regular lassos. *varlist* consists of one or more variables from *varsofinterest*. Square-root lassos are linear models, and this option cannot be used with *depvar*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `sqrtlasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. See [LASSO] **lasso options**.

The following options are available with `dslogit` but are not shown in the dialog box:

`reestimate` is an advanced option that refits the `dslogit` model based on changes made to the underlying lassos using `lassoselect`. After running `dslogit`, you can select a different λ^* for one or more of the lassos estimated by `dslogit`. After selecting λ^*, you type `dslogit, reestimate` to refit the `dslogit` model based on the newly selected λ's.

`reestimate` may be combined only with reporting options.

`noheader` prevents the coefficient table header from being displayed.

`coeflegend`; see [R] **Estimation options**.

Remarks and examples

`dslogit` performs double-selection lasso logistic regression. This command estimates odds ratios, standard errors, and confidence intervals and performs tests for variables of interest while using lassos to select from among potential control variables.

The logistic regression model is

$$\Pr(y=1|\mathbf{d},\mathbf{x}) = \frac{\exp(\mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}')}{1+\exp(\mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}')}$$

where $\mathbf{d}$ are the variables for which we wish to make inferences and $\mathbf{x}$ are the potential control variables from which the lassos select. `dslogit` estimates the $\boldsymbol{\alpha}$ coefficients and reports the corresponding odds ratios, e^{α}. However, double selection does not provide estimates of the coefficients on the control variables ($\boldsymbol{\beta}$) or their standard errors. No estimation results can be reported for $\boldsymbol{\beta}$.

For an introduction to the double-selection lasso method for inference, as well as the partialing-out and cross-fit partialing-out methods, see [LASSO] **Lasso inference intro**.

Examples that demonstrate how to use `dslogit` and the other lasso inference commands are presented in [LASSO] **Inference examples**. In particular, we recommend reading *1 Overview* for an introduction to the examples and to the `vl` command, which provides tools for working with the large lists of variables that are often included when using lassos methods. See *2 Fitting and interpreting inferential models* for comparisons of the different methods of fitting inferential models that are available in Stata. Everything we say there about methods of selection is applicable to both linear and nonlinear models. See *3 Fitting logit inferential models to binary outcomes. What is different?* for examples and discussion specific to logistic regression models. The primary difference from linear models involves interpreting the results.

If you are interested in digging deeper into the lassos that are used to select controls, see *5 Exploring inferential model lassos* in [LASSO] **Inference examples**.

Stored results

`dslogit` stores the following in `e()`:

Scalars	
`e(N)`	number of observations
`e(N_clust)`	number of clusters
`e(k_varsofinterest)`	number of variables of interest
`e(k_controls)`	number of potential control variables
`e(k_controls_sel)`	number of selected control variables
`e(df)`	degrees of freedom for test of variables of interest
`e(chi2)`	χ^2
`e(p)`	p-value for test of variables of interest
`e(rank)`	rank of `e(V)`
Macros	
`e(cmd)`	`dslogit`
`e(cmdline)`	command as typed
`e(depvar)`	name of dependent variable
`e(lasso_depvars)`	names of dependent variables for all lassos
`e(varsofinterest)`	variables of interest
`e(controls)`	potential control variables
`e(controls_sel)`	selected control variables
`e(model)`	`logit`
`e(title)`	title in estimation output
`e(offset)`	linear offset variable
`e(clustvar)`	name of cluster variable
`e(chi2type)`	`Wald`; type of χ^2 test
`e(vce)`	*vcetype* specified in `vce()`
`e(vcetype)`	title used to label Std. err.
`e(rngstate)`	random-number state used
`e(properties)`	`b V`
`e(predict)`	program used to implement `predict`
`e(select_cmd)`	program used to implement `lassoselect`
`e(marginsnotok)`	predictions disallowed by `margins`
`e(asbalanced)`	factor variables `fvset` as `asbalanced`
`e(asobserved)`	factor variables `fvset` as `asobserved`
Matrices	
`e(b)`	coefficient vector
`e(V)`	variance–covariance matrix of the estimators
Functions	
`e(sample)`	marks estimation sample

In addition to the above, the following is stored in `r()`:

Matrices	
`r(table)`	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

dslogit implements double-selection lasso logit regression (DSLLR) as described in Belloni, Chernozhukov, and Wei (2016, table 2 and sec. 2.1). The regression model is

$$\mathbf{E}[y|\mathbf{d},\,\mathbf{x}] = G(\mathbf{d}\boldsymbol{\alpha}' + \beta_0 + \mathbf{x}\boldsymbol{\beta}')$$

where $G(a) = \exp(a)/\{1 + \exp(a)\}$, $\mathbf{d}$ contains the J covariates of interest, and $\mathbf{x}$ contains the p controls. The number of covariates in $\mathbf{d}$ must be small and fixed. The number of controls in $\mathbf{x}$ can be large and, in theory, can grow with the sample size; however, the number of nonzero elements in $\boldsymbol{\beta}$ must not be too large, which is to say that the model must be sparse.

DSLLR algorithm

1. Perform a logit lasso of y on $\mathbf{d}$ and $\mathbf{x}$, and denote the selected controls by $\tilde{\mathbf{x}}$.

   This logit lasso can choose the lasso penalty parameter (λ^*) using the plugin estimator, adaptive lasso, or CV. The plugin value is the default.

2. Fit a logit regression of y on $\mathbf{d}$ and $\tilde{\mathbf{x}}$, denoting the estimated coefficient vectors by $\widetilde{\boldsymbol{\alpha}}$ and $\widetilde{\boldsymbol{\beta}}$, respectively.

3. Let $w_i = G'(\mathbf{d}_i\widetilde{\boldsymbol{\alpha}}' + \tilde{\mathbf{x}}_i\widetilde{\boldsymbol{\beta}}')$ be the ith observation of the predicted value of the derivative of $G(\cdot)$.

4. For $j = 1, \ldots, J$, perform a linear lasso of d_j on $\mathbf{x}$ using observation-level weights w_i, and denote the selected controls by $\check{\mathbf{x}}_j$.

   Each of these lassos can choose the lasso penalty parameter (λ_j^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

5. Let $\hat{\mathbf{x}}$ be the distinct variables from the union of the variables in $\check{\mathbf{x}}_1, \ldots, \check{\mathbf{x}}_J$, and $\tilde{\mathbf{x}}$.

6. Fit a logit regression of y on $\mathbf{d}$ and $\hat{\mathbf{x}}$, denoting the estimated coefficient vectors by $\widehat{\boldsymbol{\alpha}}$ and $\widehat{\boldsymbol{\beta}}$, respectively.

7. Store the point estimates $\widehat{\boldsymbol{\alpha}}$ in e(b) and their variance estimates (VCE) in e(V).

   Option vce(robust), the robust estimator of the VCE for a logistic regression, is the default. Specify option vce(oim) to get the OIM estimator of the VCE.

See *Methods and formulas* in [LASSO] **lasso** for details on how the lassos in steps 1 and 4 choose their penalty parameter (λ^*).

Reference

Belloni, A., V. Chernozhukov, and Y. Wei. 2016. Post-selection inference for generalized linear models with many controls. *Journal of Business and Economic Statistics* 34: 606–619. https://doi.org/10.1080/07350015.2016.1166116.

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **pologit** — Partialing-out lasso logistic regression

[LASSO] **xpologit** — Cross-fit partialing-out lasso logistic regression

[R] **logit** — Logistic regression, reporting coefficients

[R] **logistic** — Logistic regression, reporting odds ratios

[U] 20 Estimation and postestimation commands

dspoisson — Double-selection lasso Poisson regression

Description	Quick start	Menu	Syntax	Options
Remarks and examples	Stored results	Methods and formulas	Reference	Also see

Description

dspoisson fits a lasso Poisson regression model and reports incidence-rate ratios along with standard errors, test statistics, and confidence intervals for specified covariates of interest. The double-selection method is used to estimate effects for these variables and to select from potential control variables to be included in the model.

Quick start

Report an incidence-rate ratio from a Poisson regression of y on d1, and include x1 to x100 as potential control variables to be selected by lassos

```
dspoisson y d1, controls(x1-x100)
```

Same as above, and estimate incidence-rate ratios for the levels of categorical d2

```
dspoisson y d1 i.d2, controls(x1-x100)
```

Use cross-validation (CV) instead of a plugin iterative formula to select the optimal λ^* in each lasso

```
dspoisson y d1 i.d2, controls(x1-x100) selection(cv)
```

Same as above, and set a random-number seed for reproducibility

```
dspoisson y d1 i.d2, controls(x1-x100) selection(cv) rseed(28)
```

Specify CV for the lasso for y only, with the stopping rule criterion turned off

```
dspoisson y d1 i.d2, controls(x1-x100) lasso(y, selection(cv), stop(0))
```

Same as above, but apply the option to the lassos for y, d1, and i.d2

```
dspoisson y d1 i.d2, controls(x1-x100) lasso(*, selection(cv), stop(0))
```

Compute lassos beyond the CV minimum to get full coefficient paths, knots, etc.

```
dspoisson y d1 i.d2, controls(x1-x100) ///
   lasso(*, selection(cv, alllambdas))
```

Menu

Statistics > Lasso > Lasso inferential models > Count outcomes > Double-selection Poisson model

Syntax

```
dspoisson depvar varsofinterest [if] [in] ,
   controls([(alwaysvars)] othervars) [options]
```

varsofinterest are variables for which coefficients and their standard errors are estimated.

options	Description
Model	
* `controls([(alwaysvars)] othervars)`	*alwaysvars* and *othervars* make up the set of control variables; *alwaysvars* are always included; lassos choose whether to include or exclude *othervars*
`selection(plugin)`	use a plugin iterative formula to select an optimal value of the lasso penalty parameter λ^* for each lasso; the default
`selection(cv)`	use CV to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(adaptive)`	use adaptive lasso to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(bic)`	use BIC to select an optimal value of the lasso penalty parameter λ^* for each lasso
`sqrtlasso`	use square-root lassos for *varsofinterest*
`missingok`	after fitting lassos, ignore missing values in any *othervars* not selected, and include these observations in the final model
`offset(`$varname_o$`)`	include $varname_o$ in the lasso and model for *depvar* with its coefficient constrained to be 1
`exposure(`$varname_e$`)`	include ln($varname_e$) in the lasso and model for *depvar* with its coefficient constrained to be 1
SE/Robust	
`vce(`*vcetype*`)`	*vcetype* may be `robust` (the default), `cluster` *clustvar*, or `oim`
Reporting	
`level(#)`	set confidence level; default is `level(95)`
`irr`	report incidence-rate ratios; the default
`coef`	report estimated coefficients
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
Optimization	
`[no]log`	display or suppress an iteration log
`verbose`	display a verbose iteration log
`rseed(#)`	set random-number seed
Advanced	
`lasso(`*varlist*`,` *lasso_options*`)`	specify options for the lassos for variables in *varlist*; may be repeated
`sqrtlasso(`*varlist*`,` *lasso_options*`)`	specify options for square-root lassos for variables in *varlist*; may be repeated

`reestimate`	refit the model after using `lassoselect` to select a different λ^*
`noheader`	do not display the header on the coefficient table
`coeflegend`	display legend instead of statistics

*`controls()` is required.

varsofinterest, *alwaysvars*, and *othervars* may contain factor variables. Base levels of factor variables cannot be set for *alwaysvars* and *othervars*. See **[U] 11.4.3 Factor variables**.

`collect` is allowed; see **[U] 11.1.10 Prefix commands**.

`reestimate`, `noheader`, and `coeflegend` do not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

`controls(`[`(`*alwaysvars*`)`] *othervars*`)` specifies the set of control variables, which control for omitted variables. Control variables are also known as confounding variables. `dspoisson` fits lassos for *depvar* and each of the *varsofinterest*. *alwaysvars* are variables that are always to be included in these lassos. *alwaysvars* are optional. *othervars* are variables that each lasso will choose to include or exclude. That is, each lasso will select a subset of *othervars*. The selected subset of *othervars* may differ across lassos. `controls()` is required.

`selection(plugin` | `cv` | `adaptive` | `bic)` specifies the selection method for choosing an optimal value of the lasso penalty parameter λ^* for each lasso or square-root lasso estimation. Separate lassos are estimated for *depvar* and each variable in *varsofinterest*. Specifying `selection()` changes the selection method for all of these lassos. You can specify different selection methods for different lassos using the option `lasso()` or `sqrtlasso()`. When `lasso()` or `sqrtlasso()` is used to specify a different selection method for the lassos of some variables, they override the global setting made using `selection()` for the specified variables.

`selection(plugin)` is the default. It selects λ^* based on a "plugin" iterative formula dependent on the data. See [LASSO] **lasso options**.

`selection(cv)` selects the λ^* that gives the minimum of the CV function. See [LASSO] **lasso options**.

`selection(adaptive)` selects λ^* using the adaptive lasso selection method. It cannot be specified when `sqrtlasso` is specified. See [LASSO] **lasso options**.

`selection(bic)` selects the λ^* that gives the minimum of the BIC function. See [LASSO] **lasso options**.

`sqrtlasso` specifies that square-root lassos be done rather than regular lassos for the *varsofinterest*. This option does not apply to *depvar*. Square-root lassos are linear models, and the lasso for *depvar* is always a Poisson lasso. The option `lasso()` can be used with `sqrtlasso` to specify that regular lasso be done for some variables, overriding the global `sqrtlasso` setting for these variables. See [LASSO] **lasso options**.

`missingok` specifies that, after fitting lassos, the estimation sample be redefined based on only the nonmissing observations of variables in the final model. In all cases, any observation with missing values for *depvar*, *varsofinterest*, *alwaysvars*, and *othervars* is omitted from the estimation sample for the lassos. By default, the same sample is used for calculation of the coefficients of the *varsofinterest* and their standard errors.

When `missingok` is specified, the initial estimation sample is the same as the default, but the sample used for the calculation of the coefficients of the *varsofinterest* can be larger. Now observations with missing values for any *othervars* not selected will be added to the estimation sample (provided there are no missing values for any of the variables in the final model).

`missingok` may produce more efficient estimates when data are missing completely at random. It does, however, have the consequence that estimation samples can change when selected variables differ in models fit using different selection methods. That is, when *othervars* contain missing values, the estimation sample for a model fit using the default `selection(plugin)` will likely differ from the estimation sample for a model fit using, for example, `selection(cv)`.

`offset(`*varname*$_o$`)` specifies that *varname*$_o$ be included in the lasso and model for *depvar* with its coefficient constrained to be 1.

`exposure(`*varname*$_e$`)` specifies that ln(*varname*$_e$) be included in the lasso and model for *depvar* with its coefficient constrained to be 1.

SE/Robust

`vce(`*vcetype*`)` specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (`robust`), that allow for intragroup correlation (`vce(cluster` *clustvar*`)`), and that are derived from asymptotic theory (`vce(oim)`). See [R] ***vce_option***.

When `vce(cluster` *clustvar*`)` is specified, all lassos also account for clustering. For each lasso, this affects how the log-likelihood function is computed and how the sample is split in cross-validation; see *Methods and formulas* in [LASSO] **lasso**. Specifying `vce(cluster` *clustvar*`)` may lead to different selected controls and therefore to different point estimates for your variable of interest when compared to the estimation that ignores clustering.

Reporting

`level(#)`; see [R] **Estimation options**.

`irr` reports estimated coefficients transformed to incidence-rate ratios, that is, e^{α}. Standard errors and confidence intervals are similarly transformed. `irr` is the default.

`coef` reports the estimated coefficients α rather than the incidence-rate ratios, e^{α}. This option affects how results are displayed, not how they are estimated. `coef` may be specified at estimation or when replaying previously estimated results.

display_options: `noci`, `nopvalues`, `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `fvwrap(#)`, `fvwrapon(`*style*`)`, `cformat(%`*fmt*`)`, `pformat(%`*fmt*`)`, `sformat(%`*fmt*`)`, and `nolstretch`; see [R] **Estimation options**.

Optimization

[`no`]`log` displays or suppresses a log showing the progress of the estimation. By default, one-line messages indicating when each lasso estimation begins are shown. Specify `verbose` to see a more detailed log.

`verbose` displays a verbose log showing the iterations of each lasso estimation. This option is useful when doing `selection(cv)` or `selection(adaptive)`. It allows you to monitor the progress of the lasso estimations for these selection methods, which can be time consuming when there are many *othervars* specified in `controls()`.

`rseed(#)` sets the random-number seed. This option can be used to reproduce results for `selection(cv)` and `selection(adaptive)`. The default selection method `selection(plugin)` does not use random numbers. `rseed(#)` is equivalent to typing `set seed #` prior to running `dspoisson`. See [R] **set seed**.

Advanced

`lasso(`*varlist*`,` *lasso_options*`)` lets you set different options for different lassos, or advanced options for all lassos. You specify a *varlist* followed by the options you want to apply to the lassos for these variables. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. `_all` or `*` may be used to specify *depvar* and all *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `lasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. It also overrides the global `sqrtlasso` option for these variables. See [LASSO] **lasso options**.

`sqrtlasso(`*varlist*`,` *lasso_options*`)` works like the option `lasso()`, except square-root lassos for the variables in *varlist* are done rather than regular lassos. *varlist* consists of one or more variables from *varsofinterest*. Square-root lassos are linear models, and this option cannot be used with *depvar*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `sqrtlasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. See [LASSO] **lasso options**.

The following options are available with `dspoisson` but are not shown in the dialog box:

`reestimate` is an advanced option that refits the `dspoisson` model based on changes made to the underlying lassos using `lassoselect`. After running `dspoisson`, you can select a different λ^* for one or more of the lassos estimated by `dspoisson`. After selecting λ^*, you type `dspoisson, reestimate` to refit the `dspoisson` model based on the newly selected λ^*'s.

`reestimate` may be combined only with reporting options.

`noheader` prevents the coefficient table header from being displayed.

`coeflegend`; see [R] **Estimation options**.

Remarks and examples

`dspoisson` performs double-selection lasso Poisson regression. This command estimates incidence-rate ratios, standard errors, and confidence intervals and performs tests for variables of interest while using lassos to select from among potential control variables.

The Poisson regression model is

$$\mathbf{E}[y|\mathbf{d},\, \mathbf{x}] = \exp(\mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}')$$

where **d** are the variables for which we wish to make inferences and **x** are the potential control variables from which the lassos select. `dspoisson` estimates the $\boldsymbol{\alpha}$ coefficients and reports the corresponding incidence-rate ratios, e^{α}. However, double selection does not provide estimates of the coefficients on the control variables ($\boldsymbol{\beta}$) or their standard errors. No estimation results can be reported for $\boldsymbol{\beta}$.

For an introduction to the double-selection lasso method for inference, as well as the partialing-out and cross-fit partialing-out methods, see [LASSO] **Lasso inference intro**.

Examples that demonstrate how to use dspoisson and the other lasso inference commands are presented in [LASSO] **Inference examples**. In particular, we recommend reading *1 Overview* for an introduction to the examples and to the vl command, which provides tools for working with the large lists of variables that are often included when using lasso methods. See *2 Fitting and interpreting inferential models* for comparisons of the different methods of fitting inferential models that are available in Stata. Everything we say there about methods of selection is applicable to both linear and nonlinear models. See *4 Fitting inferential models to count outcomes. What is different?* for examples and discussion specific to Poisson regression models. The primary difference from linear models involves interpreting the results.

If you are interested in digging deeper into the lassos that are used to select controls, see *5 Exploring inferential model lassos* in [LASSO] **Inference examples**.

Stored results

dspoisson stores the following in e():

Scalars	
e(N)	number of observations
e(N_clust)	number of clusters
e(k_varsofinterest)	number of variables of interest
e(k_controls)	number of potential control variables
e(k_controls_sel)	number of selected control variables
e(df)	degrees of freedom for test of variables of interest
e(chi2)	χ^2
e(p)	p-value for test of variables of interest
e(rank)	rank of e(V)
Macros	
e(cmd)	dspoisson
e(cmdline)	command as typed
e(depvar)	name of dependent variable
e(lasso_depvars)	names of dependent variables for all lassos
e(varsofinterest)	variables of interest
e(controls)	potential control variables
e(controls_sel)	selected control variables
e(model)	poisson
e(title)	title in estimation output
e(offset)	linear offset variable
e(clustvar)	name of cluster variable
e(chi2type)	Wald; type of χ^2 test
e(vce)	*vcetype* specified in vce()
e(vcetype)	title used to label Std. err.
e(rngstate)	random-number state used
e(properties)	b V
e(predict)	program used to implement predict
e(select_cmd)	program used to implement lassoselect
e(marginsnotok)	predictions disallowed by margins
e(asbalanced)	factor variables fvset as asbalanced
e(asobserved)	factor variables fvset as asobserved
Matrices	
e(b)	coefficient vector
e(V)	variance–covariance matrix of the estimators
Functions	
e(sample)	marks estimation sample

In addition to the above, the following is stored in `r()`:

Matrices	
`r(table)`	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

`dspoisson` implements double-selection lasso Poisson regression (DSLPR) as described in Belloni, Chernozhukov, and Wei (2016, table 2 and sec. 2.1). The regression model is

$$\mathbf{E}[y|\mathbf{d},\, \mathbf{x}] = G(\mathbf{d}\boldsymbol{\alpha}' + \beta_0 + \mathbf{x}\boldsymbol{\beta}')$$

where $G(a) = \exp(a)$, $\mathbf{d}$ contains the J covariates of interest, and $\mathbf{x}$ contains the p controls. The number of covariates in $\mathbf{d}$ must be small and fixed. The number of controls in $\mathbf{x}$ can be large and, in theory, can grow with the sample size; however, the number of nonzero elements in $\boldsymbol{\beta}$ must not be too large, which is to say that the model must be sparse.

DSLPR algorithm

1. Perform a Poisson lasso of y on $\mathbf{d}$ and $\mathbf{x}$, and denote the selected controls by $\tilde{\mathbf{x}}$.

   This Poisson lasso can choose the lasso penalty parameter (λ^*) using the plugin estimator, adaptive lasso, or CV. The plugin value is the default.

2. Fit a Poisson regression of y on $\mathbf{d}$ and $\tilde{\mathbf{x}}$, denoting the estimated coefficient vectors by $\widetilde{\boldsymbol{\alpha}}$ and $\widetilde{\boldsymbol{\beta}}$, respectively.

3. Let $w_i = G'(\mathbf{d}_i\widetilde{\boldsymbol{\alpha}}' + \tilde{\mathbf{x}}_i\widetilde{\boldsymbol{\beta}}')$ be the ith observation of the predicted value of the derivative of $G(\cdot)$.

4. For $j = 1, \ldots, J$, perform a linear lasso of d_j on $\mathbf{x}$ using observation-level weights w_i, and denote the selected controls by $\check{\mathbf{x}}_j$.

   Each of these lassos can choose the lasso penalty parameter (λ_j^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

5. Let $\hat{\mathbf{x}}$ be the distinct variables from the union of the variables in $\check{\mathbf{x}}_1, \ldots, \check{\mathbf{x}}_J$, and $\tilde{\mathbf{x}}$.

6. Fit a Poisson regression of y on $\mathbf{d}$ and $\hat{\mathbf{x}}$, denoting the estimated coefficient vectors by $\widehat{\boldsymbol{\alpha}}$ and $\widehat{\boldsymbol{\beta}}$, respectively.

7. Store the point estimates $\widehat{\boldsymbol{\alpha}}$ in `e(b)` and their variance estimates (VCE) in `e(V)`.

   Option `vce(robust)`, the robust estimator of the VCE for a Poisson regression, is the default. Specify option `vce(oim)` to get the OIM estimator of the VCE.

See *Methods and formulas* in [LASSO] **lasso** for details on how the lassos in steps 1 and 4 choose their penalty parameter (λ^*).

Reference

Belloni, A., V. Chernozhukov, and Y. Wei. 2016. Post-selection inference for generalized linear models with many controls. *Journal of Business and Economic Statistics* 34: 606–619. https://doi.org/10.1080/07350015.2016.1166116.

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **popoisson** — Partialing-out lasso Poisson regression

[LASSO] **xpopoisson** — Cross-fit partialing-out lasso Poisson regression

[R] **poisson** — Poisson regression

[U] 20 Estimation and postestimation commands

dsregress — Double-selection lasso linear regression

Description

`dsregress` fits a lasso linear regression model and reports coefficients along with standard errors, test statistics, and confidence intervals for specified covariates of interest. The double-selection method is used to estimate effects for these variables and to select from potential control variables to be included in the model.

Quick start

Estimate a coefficient for `d1` in a linear regression of `y` on `d1`, and include `x1` to `x100` as potential control variables to be selected by lassos

```
dsregress y d1, controls(x1-x100)
```

Same as above, and estimate coefficients for the levels of categorical `d2`

```
dsregress y d1 i.d2, controls(x1-x100)
```

Use cross-validation (CV) instead of a plugin iterative formula to select the optimal λ^* in each lasso

```
dsregress y d1 i.d2, controls(x1-x100) selection(cv)
```

Same as above, and set a random-number seed for reproducibility

```
dsregress y d1 i.d2, controls(x1-x100) selection(cv) rseed(28)
```

Specify CV for the lasso for `y` only, with the stopping rule criterion turned off

```
dsregress y d1 i.d2, controls(x1-x100) lasso(y, selection(cv), stop(0))
```

Same as above, but apply the option to the lassos for `y`, `d1`, and `i.d2`

```
dsregress y d1 i.d2, controls(x1-x100) lasso(*, selection(cv), stop(0))
```

Compute lassos beyond the CV minimum to get full coefficient paths, knots, etc.

```
dsregress y d1 i.d2, controls(x1-x100) ///
   lasso(*, selection(cv, alllambdas))
```

Menu

Statistics > Lasso > Lasso inferential models > Continuous outcomes > Double-selection model

Syntax

dsregress *depvar varsofinterest* [*if*] [*in*] ,
    controls([(*alwaysvars*)] *othervars*) [*options*]

varsofinterest are variables for which coefficients and their standard errors are estimated.

options	Description
Model	
* controls([(*alwaysvars*)] *othervars*)	*alwaysvars* and *othervars* make up the set of control variables; *alwaysvars* are always included; lassos choose whether to include or exclude *othervars*
selection(plugin)	use a plugin iterative formula to select an optimal value of the lasso penalty parameter λ^* for each lasso; the default
selection(cv)	use CV to select an optimal value of the lasso penalty parameter λ^* for each lasso
selection(adaptive)	use adaptive lasso to select an optimal value of the lasso penalty parameter λ^* for each lasso
selection(bic)	use BIC to select an optimal value of the lasso penalty parameter λ^* for each lasso
sqrtlasso	use square-root lassos
missingok	after fitting lassos, ignore missing values in any *othervars* not selected, and include these observations in the final model
SE/Robust	
vce(*vcetype*)	*vcetype* may be robust (the default), cluster *clustvar*, ols, hc2, or hc3
Reporting	
level(#)	set confidence level; default is level(95)
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
Optimization	
[no]log	display or suppress an iteration log
verbose	display a verbose iteration log
rseed(#)	set random-number seed
Advanced	
lasso(*varlist*, *lasso_options*)	specify options for the lassos for variables in *varlist*; may be repeated
sqrtlasso(*varlist*, *lasso_options*)	specify options for square-root lassos for variables in *varlist*; may be repeated
reestimate	refit the model after using lassoselect to select a different λ^*
noheader	do not display the header on the coefficient table
coeflegend	display legend instead of statistics

*controls() is required.

varsofinterest, *alwaysvars*, and *othervars* may contain factor variables. Base levels of factor variables cannot be set for *alwaysvars* and *othervars*. See **[U] 11.4.3 Factor variables**.

collect is allowed; see **[U] 11.1.10 Prefix commands**.

reestimate, noheader, and coeflegend do not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

controls([(*alwaysvars*)] *othervars*) specifies the set of control variables, which control for omitted variables. Control variables are also known as confounding variables. dsregress fits lassos for *depvar* and each of the *varsofinterest*. *alwaysvars* are variables that are always to be included in these lassos. *alwaysvars* are optional. *othervars* are variables that each lasso will choose to include or exclude. That is, each lasso will select a subset of *othervars*. The selected subset of *othervars* may differ across lassos. controls() is required.

selection(plugin | cv | adaptive | bic) specifies the selection method for choosing an optimal value of the lasso penalty parameter λ^* for each lasso or square-root lasso estimation. Separate lassos are estimated for *depvar* and each variable in *varsofinterest*. Specifying selection() changes the selection method for all of these lassos. You can specify different selection methods for different lassos using the option lasso() or sqrtlasso(). When lasso() or sqrtlasso() is used to specify a different selection method for the lassos of some variables, they override the global setting made using selection() for the specified variables.

selection(plugin) is the default. It selects λ^* based on a "plugin" iterative formula dependent on the data. See [LASSO] **lasso options**.

selection(cv) selects the λ^* that gives the minimum of the CV function. See [LASSO] **lasso options**.

selection(adaptive) selects λ^* using the adaptive lasso selection method. It cannot be specified when sqrtlasso is specified. See [LASSO] **lasso options**.

selection(bic) selects the λ^* that gives the minimum of the BIC function. See [LASSO] **lasso options**.

sqrtlasso specifies that square-root lassos be done rather than regular lassos. The option lasso() can be used with sqrtlasso to specify that regular lasso be done for some variables, overriding the global sqrtlasso setting for these variables. See [LASSO] **lasso options**.

missingok specifies that, after fitting lassos, the estimation sample be redefined based on only the nonmissing observations of variables in the final model. In all cases, any observation with missing values for *depvar*, *varsofinterest*, *alwaysvars*, and *othervars* is omitted from the estimation sample for the lassos. By default, the same sample is used for calculation of the coefficients of the *varsofinterest* and their standard errors.

When missingok is specified, the initial estimation sample is the same as the default, but the sample used for the calculation of the coefficients of the *varsofinterest* can be larger. Now observations with missing values for any *othervars* not selected will be added to the estimation sample (provided there are no missing values for any of the variables in the final model).

`missingok` may produce more efficient estimates when data are missing completely at random. It does, however, have the consequence that estimation samples can change when selected variables differ in models fit using different selection methods. That is, when *othervars* contain missing values, the estimation sample for a model fit using the default `selection(plugin)` will likely differ from the estimation sample for a model fit using, for example, `selection(cv)`.

SE/Robust

`vce(`*vcetype*`)` specifies the type of standard error reported. The default is `vce(robust)`, which is robust to some kinds of misspecification. Also available are `vce(cluster` *clustvar*`)`, which allows for intragroup correlation; `vce(ols)`, which specifies the standard variance estimator for ordinary least-squares regression; and `vce(hc2)` and `vce(hc3)`, which specify alternative bias corrections for the robust variance calculation. See [R] ***vce_option*** and *Options* in [R] **regress**.

When `vce(cluster` *clustvar*`)` is specified, all lassos also account for clustering. For each lasso, this affects how the log-likelihood function is computed and how the sample is split in cross-validation; see *Methods and formulas* in [LASSO] **lasso**. Specifying `vce(cluster` *clustvar*`)` may lead to different selected controls and therefore to different point estimates for your variable of interest when compared to the estimation that ignores clustering.

Reporting

`level(#)`; see [R] **Estimation options**.

display_options: `noci`, `nopvalues`, `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `fvwrap(#)`, `fvwrapon(`*style*`)`, `cformat(%`*fmt*`)`, `pformat(%`*fmt*`)`, `sformat(%`*fmt*`)`, and `nolstretch`; see [R] **Estimation options**.

Optimization

[`no`]`log` displays or suppresses a log showing the progress of the estimation. By default, one-line messages indicating when each lasso estimation begins are shown. Specify `verbose` to see a more detailed log.

`verbose` displays a verbose log showing the iterations of each lasso estimation. This option is useful when doing `selection(cv)` or `selection(adaptive)`. It allows you to monitor the progress of the lasso estimations for these selection methods, which can be time consuming when there are many *othervars* specified in `controls()`.

`rseed(#)` sets the random-number seed. This option can be used to reproduce results for `selection(cv)` and `selection(adaptive)`. The default selection method `selection(plugin)` does not use random numbers. `rseed(#)` is equivalent to typing `set seed #` prior to running `dsregress`. See [R] **set seed**.

Advanced

`lasso(`*varlist*`,` *lasso_options*`)` lets you set different options for different lassos, or advanced options for all lassos. You specify a *varlist* followed by the options you want to apply to the lassos for these variables. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. `_all` or `*` may be used to specify *depvar* and all *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `lasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. It also overrides the global `sqrtlasso` option for these variables. See [LASSO] **lasso options**.

`sqrtlasso(`*varlist*`,` *lasso_options*`)` works like the option `lasso()`, except square-root lassos for the variables in *varlist* are done rather than regular lassos. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `sqrtlasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. See [LASSO] **lasso options**.

The following options are available with `dsregress` but are not shown in the dialog box:

`reestimate` is an advanced option that refits the `dsregress` model based on changes made to the underlying lassos using `lassoselect`. After running `dsregress`, you can select a different λ^* for one or more of the lassos estimated by `dsregress`. After selecting λ^*, you type `dsregress, reestimate` to refit the `dsregress` model based on the newly selected λ's.

`reestimate` may be combined only with reporting options.

`noheader` prevents the coefficient table header from being displayed.

`coeflegend`; see [R] **Estimation options**.

Remarks and examples

`dsregress` performs double-selection lasso linear regression. This command estimates coefficients, standard errors, and confidence intervals and performs tests for variables of interest while using lassos to select from among potential control variables.

The linear regression model is

$$\mathbf{E}[y|\mathbf{d},\,\mathbf{x}] = \mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}'$$

where **d** are the variables for which we wish to make inferences and **x** are the potential control variables from which the lassos select. `dsregress` reports estimated coefficients for $\boldsymbol{\alpha}$. However, double-selection does not provide estimates of the coefficients on the control variables ($\boldsymbol{\beta}$) or their standard errors. No estimation results can be reported for $\boldsymbol{\beta}$.

For an introduction to the double-selection lasso method for inference, as well as the partialing-out and cross-fit partialing-out methods, see [LASSO] **Lasso inference intro**.

Examples that demonstrate how to use `dsregress` and the other lasso inference commands are presented in [LASSO] **Inference examples**. In particular, we recommend reading *1 Overview* for an introduction to the examples and to the `vl` command, which provides tools for working with the large lists of variables that are often included when using lasso methods. See *2 Fitting and interpreting inferential models* for examples of fitting inferential lasso linear models and comparisons of the different methods available in Stata.

If you are interested in digging deeper into the lassos that are used to select controls, see *5 Exploring inferential model lassos* in [LASSO] **Inference examples**.

Stored results

dsregress stores the following in e():

Scalars	
e(N)	number of observations
e(N_clust)	number of clusters
e(k_varsofinterest)	number of variables of interest
e(k_controls)	number of potential control variables
e(k_controls_sel)	number of selected control variables
e(df)	degrees of freedom for test of variables of interest
e(chi2)	χ^2
e(p)	p-value for test of variables of interest
e(rank)	rank of e(V)
Macros	
e(cmd)	dsregress
e(cmdline)	command as typed
e(depvar)	name of dependent variable
e(lasso_depvars)	names of dependent variables for all lassos
e(varsofinterest)	variables of interest
e(controls)	potential control variables
e(controls_sel)	selected control variables
e(model)	linear
e(title)	title in estimation output
e(clustvar)	name of cluster variable
e(chi2type)	Wald; type of χ^2 test
e(vce)	*vcetype* specified in vce()
e(vcetype)	title used to label Std. err.
e(rngstate)	random-number state used
e(properties)	b V
e(predict)	program used to implement predict
e(select_cmd)	program used to implement lassoselect
e(marginsnotok)	predictions disallowed by margins
e(asbalanced)	factor variables fvset as asbalanced
e(asobserved)	factor variables fvset as asobserved
Matrices	
e(b)	coefficient vector
e(V)	variance–covariance matrix of the estimators
Functions	
e(sample)	marks estimation sample

In addition to the above, the following is stored in r():

Matrices	
r(table)	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in r() are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

dsregress implements double-selection lasso as described in Belloni, Chernozhukov, and Hansen (2014). The regression model is

$$\mathbf{E}[y|\mathbf{d},\, \mathbf{x}] = \mathbf{d}\boldsymbol{\alpha}' + \beta_0 + \mathbf{x}\boldsymbol{\beta}'$$

where $\mathbf{d}$ contains the J covariates of interest and $\mathbf{x}$ contains the p controls. The number of covariates in $\mathbf{d}$ must be small and fixed. The number of controls in $\mathbf{x}$ can be large and, in theory, can grow with the sample size; however, the number of nonzero elements in $\boldsymbol{\beta}$ must not be too large, which is to say that the model must be sparse.

Double-selection lasso algorithm

1. Perform a linear lasso of y on $\mathbf{x}$, and denote the selected variables by $\tilde{\mathbf{x}}_y$.

   This lasso can choose the lasso penalty parameter (λ^*) using the plugin estimator, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

2. For $j = 1, \dots, J$, perform a linear lasso of d_j on $\mathbf{x}$, and denote the selected controls by $\tilde{\mathbf{x}}_j$.

   Each of these lassos can choose the lasso penalty parameter (λ_j^*) using the plugin estimator, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

3. Let $\hat{\mathbf{x}}$ be the distinct variables in the union of the variables in $\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_J$, and $\tilde{\mathbf{x}}_y$.

4. Fit a linear regression of y on $\mathbf{d}$ and $\hat{\mathbf{x}}$, denoting the estimated coefficient vectors by $\widehat{\boldsymbol{\alpha}}$ and $\widehat{\boldsymbol{\beta}}$, respectively.

5. Store the point estimates $\widehat{\boldsymbol{\alpha}}$ in `e(b)` and their variance estimates (VCE) in `e(V)`.

   Option `vce(robust)`, the robust estimator of the VCE for a linear regression, is the default. See *Methods and formulas* in [R] **regress** for details about option `vce(robust)` and the other VCE estimators available via options `vce(ols)`, `vce(hc2)`, and `vce(hc3)`.

See *Methods and formulas* in [LASSO] **lasso** for details on how the lassos in steps 1 and 2 choose their penalty parameter (λ^*).

Reference

Belloni, A., V. Chernozhukov, and C. B. Hansen. 2014. Inference on treatment effects after selection among high-dimensional controls. *Review of Economic Studies* 81: 608–650. https://doi.org/10.1093/restud/rdt044.

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **poregress** — Partialing-out lasso linear regression

[LASSO] **xporegress** — Cross-fit partialing-out lasso linear regression

[R] **regress** — Linear regression

[U] 20 Estimation and postestimation commands

elasticnet — Elastic net for prediction and model selection

Description	Quick start	Menu	Syntax	Options
Remarks and examples	Stored results	Methods and formulas	References	Also see

Description

`elasticnet` selects covariates and fits linear, logistic, probit, Poisson, and Cox proportional hazards models using elastic net. Results from `elasticnet` can be used for prediction and model selection.

`elasticnet` saves but does not display estimated coefficients. The postestimation commands listed in [LASSO] **lasso postestimation** can be used to generate predictions, report coefficients, and display measures of fit.

For an introduction to lasso, see [LASSO] **Lasso intro**.

Quick start

Fit a linear model for `y1`, and select covariates from `x1` to `x100` using cross-validation (CV)

```
elasticnet linear y1 x1-x100
```

Same as above, but specify the grid $\alpha = 0.1, 0.2, \ldots, 1$ using a numlist

```
elasticnet linear y1 x1-x100, alpha(0.1(0.1)1)
```

Same as above, but force `x1` and `x2` to be in the model while `elasticnet` selects `x3` to `x100`

```
elasticnet linear y1 (x1 x2) x3-x100, alpha(0.1(0.1)1)
```

Fit a logistic model for binary outcome `y2` with grid $\alpha = 0.7, 0.8, 0.9, 1$

```
elasticnet logit y2 x1-x100, alpha(0.7 0.8 0.9 1)
```

Same as above, and set a random-number seed for reproducibility

```
elasticnet logit y2 x1-x100, alpha(0.7 0.8 0.9 1) rseed(1234)
```

Fit a Poisson model for count outcome `y3` with exposure `time`

```
elasticnet poisson y3 x1-x100, alpha(0.1(0.1)1) exposure(time)
```

Calculate the CV function beyond the CV minimum to get the full coefficient paths, knots, etc.

```
elasticnet linear y1 x1-x100, alpha(0.1(0.1)1) selection(cv, alllambdas)
```

Turn off the early stopping rule, and iterate over λ's until a minimum is found or until the end of the λ grid is reached

```
elasticnet linear y1 x1-x100, alpha(0.1(0.1)1) stop(0)
```

Fit a Cox proportional hazards model for `t` with failure indicator `fail`, and select covariates from `x1` to `x100` using CV

```
stset t, failure(fail)
elasticnet cox x1-x100
```

Same as above, but select covariates by minimizing the BIC

```
elasticnet cox x1-x100, selection(bic)
```

Menu

Statistics > Lasso > Elastic net

Syntax

For linear, logit, probit, and Poisson models

`elasticnet` *model* *depvar* [(*alwaysvars*)] *othervars* [*if*] [*in*] [*weight*] [, *options*]

For Cox models

`elasticnet cox` [(*alwaysvars*)] *othervars* [*if*] [*in*] [, *options*]

model is one of `linear`, `logit`, `probit`, or `poisson`.

alwaysvars are variables that are always included in the model.

othervars are variables that `elasticnet` will choose to include in or exclude from the model.

options	Description
Model	
* `noconstant`	suppress constant term
`selection(cv` [, *cv_opts*]`)`	select mixing parameter α^* and lasso penalty parameter λ^* using CV
`selection(bic` [, *bic_opts*]`)`	select mixing parameter α^* and lasso penalty parameter λ^* using BIC
`selection(none)`	do not select α^* or λ^*
`offset(`*varname*$_o$`)`	include *varname*$_o$ in model with coefficient constrained to 1
`exposure(`*varname*$_e$`)`	include ln(*varname*$_e$) in model with coefficient constrained to 1 (`poisson` model only)
* `cluster(`*clustvar*`)`	specify cluster variable *clustvar*
Optimization	
[`no`]`log`	display or suppress an iteration log
`rseed(#)`	set random-number seed
`alphas(`*numlist* | *matname*`)`	specify the α grid with *numlist* or a matrix
`grid(#`$_g$ [, `ratio(#) min(#)`]`)`	specify the set of possible λ's using a logarithmic grid with #$_g$ grid points
`crossgrid(augmented)`	augment the λ grids for each α as necessary to produce a single λ grid; the default
`crossgrid(union)`	use the union of the λ grids for each α to produce a single λ grid
`crossgrid(different)`	use different λ grids for each α
`stop(#)`	tolerance for stopping the iteration over the λ grid early
`cvtolerance(#)`	tolerance for identification of the CV function minimum
`bictolerance(#)`	tolerance for identification of the BIC function minimum
`tolerance(#)`	convergence tolerance for coefficients based on their values
`dtolerance(#)`	convergence tolerance for coefficients based on deviance
`penaltywt(`*matname*`)`	programmer's option for specifying a vector of weights for the coefficients in the penalty term

cv_opts	Description
`folds(#)`	use # folds for CV
`alllambdas`	fit models for all λ's in the grid or until the `stop(#)` tolerance is reached; by default, the CV function is calculated sequentially by λ, and estimation stops when a minimum is identified
`serule`	use the one-standard-error rule to select λ^*
`stopok`	when, for a value of α, the CV function does not have an identified minimum and the `stop(#)` stopping criterion for λ was reached at λ_{stop}, allow λ_{stop} to be included in an (α, λ) pair that can potentially be selected as (α^*, λ^*); the default
`strict`	requires the CV function to have an identified minimum for every value of α; this is a stricter alternative to the default `stopok`
`gridminok`	when, for a value of α, the CV function does not have an identified minimum and the `stop(#)` stopping criterion for λ was not reached, allow the minimum of the λ grid, λ_{gmin}, to be included in an (α, λ) pair that can potentially be selected as (α^*, λ^*); this is a looser alternative to the default `stopok` and is rarely used

bic_opts	Description
`alllambdas`	fit models for all λ's in the grid or until the `stop(#)` tolerance is reached; by default, the BIC function is calculated sequentially by λ, and estimation stops when a minimum is identified
`stopok`	when, for a value of α, the BIC function does not have an identified minimum and the `stop(#)` stopping criterion for λ was reached at λ_{stop}, allow λ_{stop} to be included in an (α, λ) pair that can potentially be selected as (α^*, λ^*); the default
`strict`	requires the BIC function to have an identified minimum for every value of α; this is a stricter alternative to the default `stopok`
`gridminok`	when, for a value of α, the BIC function does not have an identified minimum and the `stop(#)` stopping criterion for λ was not reached, allow the minimum of the λ grid, λ_{gmin}, to be included in an (α, λ) pair that can potentially be selected as (α^*, λ^*); this is a looser alternative to the default `stopok` and is rarely used
`postselection`	use postselection coefficients to compute BIC

*`noconstant` and `cluster()` are not allowed with `elasticnet cox`.

You must `stset` your data before using `elasticnet cox`; see [ST] **stset**.

alwaysvars and *othervars* may contain factor variables; see **[U] 11.4.3 Factor variables**.

`collect` is allowed; see **[U] 11.1.10 Prefix commands**.

Default weights are not allowed. `iweight`s are allowed with all *sel_method* options. `fweight`s are allowed when `selection(plugin)`, `selection(bic)`, or `selection(none)` is specified. See **[U] 11.1.6 weight**. For `elasticnet cox`, weights must be specified when you `stset` your data.

`penaltywt(`*matname*`)` does not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

See [LASSO] **lasso fitting** for an overview of the lasso estimation procedure and a detailed description of how to set options to control it.

Model

noconstant omits the constant term. Note, however, when there are factor variables among the *othervars*, elasticnet can potentially create the equivalent of the constant term by including all levels of a factor variable. This option is likely best used only when all the *othervars* are continuous variables and there is a conceptual reason why there should be no constant term. This option is not allowed with cox.

selection(cv), selection(bic), and selection(none) specify the selection method used to select λ^*.

selection(cv [, *cv_opts*]) is the default. It selects the (α^*, λ^*) that give the minimum of the CV function.

folds(#) specifies that CV with # folds be done. The default is folds(10).

alllambdas specifies that, for each α, models be fit for all λ's in the grid or until the stop(#) tolerance is reached. By default, models are calculated sequentially from largest to smallest λ, and the CV function is calculated after each model is fit. If a minimum of the CV function is found, the computation ends at that point without evaluating additional smaller λ's.

alllambdas computes models for these additional smaller λ's. Because computation time is greater for smaller λ, specifying alllambdas may increase computation time manyfold. Specifying alllambdas is typically done only when a full plot of the CV function is wanted for assurance that a true minimum has been found. Regardless of whether alllambdas is specified, the selected (α^*, λ^*) will be the same.

serule selects λ^* based on the "one-standard-error rule" recommended by Hastie, Tibshirani, and Wainwright (2015, 13–14) instead of the λ that minimizes the CV function. The one-standard-error rule selects, for each α, the largest λ for which the CV function is within a standard error of the minimum of the CV function. Then, from among these (α, λ) pairs, the one with the smallest value of the CV function is selected.

stopok, strict, and gridminok specify what to do when, for a value of α, the CV function does not have an identified minimum at any value of λ in the grid. A minimum is identified at λ_{cvmin} when the CV function at both larger and smaller adjacent λ's is greater than it is at λ_{cvmin}. When the CV function for a value of α has an identified minimum, these options all do the same thing: $(\alpha, \lambda_{\text{cvmin}})$ becomes one of the (α, λ) pairs that potentially can be selected as the smallest value of the CV function. In some cases, however, the CV function declines monotonically as λ gets smaller and never rises to identify a minimum. When the CV function does not have an identified minimum, stopok and gridminok make alternative picks for λ in the (α, λ) pairs that will be assessed for the smallest value of the CV function. The option strict makes no alternative pick for λ. You may specify only one of stopok, strict, or gridminok; stopok is the default if you do not specify one. With each of these options, estimation results are always left in place, and alternative (α, λ) pairs can be selected and evaluated.

stopok specifies that, for a value of α, when the CV function does not have an identified minimum and the stop(#) stopping tolerance for λ was reached at λ_{stop}, the pair $(\alpha, \lambda_{\text{stop}})$ is picked as one of the pairs that potentially can be selected as the smallest value of the CV function. λ_{stop} is the smallest λ for which coefficients are estimated, and it is assumed that λ_{stop} has a CV function value close to the true minimum for that value of α. When no minimum is identified for a value of α and the stop(#) criterion is not met, an error is issued.

strict requires the CV function to have an identified minimum for each value of α, and if not, an error is issued.

gridminok is a rarely used option that specifies that, for a value of α, when the CV function has no identified minimum and the stop(#) stopping criterion was not met, λ_{gmin}, the minimum of the λ grid, is picked as part of a pair $(\alpha, \lambda_{\text{gmin}})$ that potentially can be selected as the smallest value of the CV function.

The gridminok criterion is looser than the default stopok, which is looser than strict. With strict, the selected (α^*, λ^*) pair is the minimum of the CV function chosen from the $(\alpha, \lambda_{\text{cvmin}})$ pairs, where all λ's under consideration are identified minimums. With stopok, the set of (α, λ) pairs under consideration for the minimum of the CV function include identified minimums, λ_{cvmin}, or values, λ_{stop}, that met the stopping criterion. With gridminok, the set of (α, λ) pairs under consideration for the minimum of the CV function potentially include λ_{cvmin}, λ_{stop}, or λ_{gmin}.

selection(bic [, *bic_opts*]) selects (α^*, λ^*) by using the Bayesian information criterion (BIC) function. It selects the (α^*, λ^*) with the minimum BIC function value.

bic_opts are alllambdas, stopok, strict, gridminok, and postselection.

alllambdas specifies that, for each α, models be fit for all λ's in the grid or until the stop(#) tolerance is reached. By default, models are calculated sequentially from largest to smallest λ, and the BIC function is calculated after each model is fit. If a minimum of the BIC function is found, the computation ends at that point without evaluating additional smaller λ's.

alllambdas computes models for these additional smaller λ's. Because computation time is greater for smaller λ, specifying alllambdas may increase computation time manyfold. Specifying alllambdas is typically done only when a full plot of the BIC function is wanted for assurance that a true minimum has been found. Regardless of whether alllambdas is specified, the selected (α^*, λ^*) will be the same.

stopok, strict, and gridminok specify what to do when, for a value of α, the BIC function does not have an identified minimum at any value of λ in the grid. A minimum is identified at λ_{bicmin} when the BIC function at both larger and smaller adjacent λ's is greater than it is at λ_{bicmin}. When the BIC function for a value of α has an identified minimum, these options all do the same thing: $(\alpha, \lambda_{\text{bicmin}})$ becomes one of the (α, λ) pairs that potentially can be selected as the smallest value of the BIC function. In some cases, however, the BIC function declines monotonically as λ gets smaller and never rises to identify a minimum. When the BIC function does not have an identified minimum, stopok and gridminok make alternative picks for λ in the (α, λ) pairs that will be assessed for the smallest value of the BIC function. The option strict makes no alternative pick for λ. You may specify only one of stopok, strict, or gridminok; stopok is the default if you do not specify one. With each of these options, estimation results are always left in place, and alternative (α, λ) pairs can be selected and evaluated.

stopok specifies that, for a value of α, when the BIC function does not have an identified minimum and the stop(#) stopping tolerance for λ was reached at λ_{stop}, the pair $(\alpha, \lambda_{\text{stop}})$ is picked as one of the pairs that potentially can be selected as the smallest value of the BIC function. λ_{stop} is the smallest λ for which coefficients are estimated, and it is assumed that λ_{stop} has a BIC function value close to the true minimum for that value of α. When no minimum is identified for a value of α and the stop(#) criterion is not met, an error is issued.

strict requires the BIC function to have an identified minimum for each value of α, and if not, an error is issued.

`gridminok` is a rarely used option that specifies that, for a value of α, when the BIC function has no identified minimum and the `stop(#)` stopping criterion was not met, λ_{gmin}, the minimum of the λ grid, is picked as part of a pair $(\alpha, \lambda_{\text{gmin}})$ that potentially can be selected as the smallest value of the BIC function.

The `gridminok` criterion is looser than the default `stopok`, which is looser than `strict`. With `strict`, the selected (α^*, λ^*) pair is the minimum of the BIC function chosen from the $(\alpha, \lambda_{\text{bicmin}})$ pairs, where all λ's under consideration are identified minimums. With `stopok`, the set of (α, λ) pairs under consideration for the minimum of the BIC function include identified minimums, λ_{bicmin}, or values, λ_{stop}, that met the stopping criterion. With `gridminok`, the set of (α, λ) pairs under consideration for the minimum of the BIC function potentially include λ_{bicmin}, λ_{stop}, or λ_{gmin}.

`postselection` specifies to use the postselection coefficients to compute the BIC function. By default, the penalized coefficients are used.

`selection(none)` does not select an (α^*, λ^*) pair. In this case, the elastic net is estimated for a grid of values for λ for each α, but no attempt is made to determine which (α, λ) pair is best. The postestimation command `lassoknots` can be run to view a table of λ's that define the knots (that is, the distinct sets of nonzero coefficients) for each α. The `lassoselect` command can then be used to select an (α^*, λ^*) pair, and `lassogof` can be run to evaluate the prediction performance of the selected pair.

When `selection(none)` is specified, neither the CV function nor the BIC function is computed. If you want to view the knot table with values of the CV function shown and then select (α^*, λ^*), you must specify `selection(cv)`. Similarly, if you want to view the knot table with values of the BIC function shown, you must specify `selection(bic)`. There are no suboptions for `selection(none)`.

`offset(`*varname*$_o$`)` specifies that *varname*$_o$ be included in the model with its coefficient constrained to be 1.

`exposure(`*varname*$_e$`)` can be specified only for the `poisson` model. It specifies that ln(*varname*$_e$) be included in the model with its coefficient constrained to be 1.

`cluster(`*clustvar*`)` specifies the cluster variable *clustvar*. Specifying a cluster variable will affect how the log-likelihood function is computed and the sample split in cross-validation. The log-likelihood function is computed as the sum of the log likelihood at the cluster levels. If option `selection(cv)` is specified, the cross-validation sample is split by the clusters defined by *clustvar*. That is, the subsample in each fold is drawn on the cluster level. Therefore, all observations in a cluster are kept together in the same subsample. This option is not allowed with `elasticnet cox`.

Optimization

`[no]log` displays or suppresses a log showing the progress of the estimation.

`rseed(#)` sets the random-number seed. This option can be used to reproduce results for `selection(cv)`. (`selection(bic)` and `selection(none)` do not use random numbers.) `rseed(#)` is equivalent to typing `set seed #` prior to running `elasticnet`. See [R] **set seed**.

`alphas(`*numlist* | *matname*`)` specifies either a numlist or a matrix containing the grid of values for α. The default is `alphas(0.5 0.75 1)`. Specifying a small, nonzero value of α for one of the values of `alphas()` will result in lengthy computation time because the optimization algorithm for a penalty that is mostly ridge regression with a little lasso mixed in is inherently inefficient. Pure ridge regression ($\alpha = 0$), however, is computationally efficient.

`grid(`$\#_g$ [`, ratio(#) min(#)`]`)` specifies the set of possible λ's using a logarithmic grid with $\#_g$ grid points.

$\#_g$ is the number of grid points for λ. The default is $\#_g = 100$. The grid is logarithmic with the ith grid point ($i = 1, \dots, n = \#_g$) given by $\ln \lambda_i = [(i-1)/(n-1)] \ln r + \ln \lambda_{\text{gmax}}$, where $\lambda_{\text{gmax}} = \lambda_1$ is the maximum, $\lambda_{\text{gmin}} = \lambda_n =$ `min(#)` is the minimum, and $r = \lambda_{\text{gmin}}/\lambda_{\text{gmax}} =$ `ratio(#)` is the ratio of the minimum to the maximum.

`ratio(#)` specifies $\lambda_{\text{gmin}}/\lambda_{\text{gmax}}$. The maximum of the grid, λ_{gmax}, is set to the smallest λ for which all the coefficients in the lasso are estimated to be zero (except the coefficients of the *alwaysvars*). λ_{gmin} is then set based on `ratio(#)`. When $p < N$, where p is the total number of *othervars* and *alwaysvars* (not including the constant term) and N is the number of observations, the default value of `ratio(#)` is 1e−4. When $p \geq N$, the default is 1e−2.

`min(#)` sets λ_{gmin}. By default, λ_{gmin} is based on `ratio(#)` and λ_{gmax}, which is computed from the data.

`crossgrid(augmented)`, `crossgrid(union)`, and `crossgrid(different)` specify the type of two-dimensional grid used for (α, λ). `crossgrid(augmented)` and `crossgrid(union)` produce a grid that is the product of two one-dimensional grids. That is, the λ grid is the same for every value of α. `crossgrid(different)` uses different λ grids for different values of α

`crossgrid(augmented)`, the default grid, is formed by an augmentation algorithm. First, a suitable λ grid for each α is computed. Then, nonoverlapping segments of these grids are formed and combined into a single λ grid.

`crossgrid(union)` specifies that the union of λ grids across each value of α be used. That is, a λ grid for each α is computed, and then they are combined by simply putting all the λ values into one grid that is used for each α. This produces a fine grid that can cause the computation to take a long time without significant gain in most cases.

`crossgrid(different)` specifies that different λ grids be used for each value of α. This option is rarely used. Using different λ grids for different values of α complicates the interpretation of the CV selection method. When the λ grid is not the same for every value of α, comparisons are based on parameter intervals that are not on the same scale.

`stop(#)` specifies a tolerance that is the stopping criterion for the λ iterations. The default is 1e−5. Estimation starts with the maximum grid value, λ_{gmax}, and iterates toward the minimum grid value, λ_{gmin}. When the relative difference in the deviance produced by two adjacent λ grid values is less than `stop(#)`, the iteration stops and no smaller λ's are evaluated. The value of λ that meets this tolerance is denoted by λ_{stop}. Typically, this stopping criterion is met before the iteration reaches λ_{gmin}.

Setting `stop(#)` to a larger value means that iterations are stopped earlier at a larger λ_{stop}. To produce coefficient estimates for all values of the λ grid, you can specify `stop(0)`. Note, however, that computations for small λ's can be extremely time consuming. In terms of time, when you use `selection(cv)` or `selection(bic)`, the optimal value of `stop(#)` is the largest value that allows estimates for just enough λ's to be computed to identify the minimum of the CV or BIC function. When setting `stop(#)` to larger values, be aware of the consequences of the default λ^* selection procedure given by the default `stopok`. You may want to override the `stopok` behavior by using `strict`.

`cvtolerance(#)` is a rarely used option that changes the tolerance for identifying the minimum CV function. For linear models, a minimum is identified when the CV function rises above a nominal minimum for at least three smaller λ's with a relative difference in the CV function greater than #. For nonlinear models, at least five smaller λ's are required. The default is 1e−3. Setting # to a bigger value makes a stricter criterion for identifying a minimum and brings more assurance that a declared minimum is a true minimum, but it also means that models may need to be fit for additional smaller λ, which can be time consuming. See *Methods and formulas* for [LASSO] **lasso** for more information about this tolerance and the other tolerances.

`bictolerance(#)` is a rarely used option that changes the tolerance for identifying the minimum BIC function. A minimum is identified when the BIC function rises above a nominal minimum for at least two smaller λ's with a relative difference in the BIC function greater than #. The default is 1e−2. Setting # to a bigger value makes a stricter criterion for identifying a minimum and brings more assurance that a declared minimum is a true minimum, but it also means that models may need to be fit for additional smaller λ, which can be time consuming. See *Methods and formulas* in [LASSO] **lasso** for more information about this tolerance and the other tolerances.

`tolerance(#)` is a rarely used option that specifies the convergence tolerance for the coefficients. Convergence is achieved when the relative change in each coefficient is less than this tolerance. The default is `tolerance(1e-7)`.

`dtolerance(#)` is a rarely used option that changes the convergence criterion for the coefficients. When `dtolerance(#)` is specified, the convergence criterion is based on the change in deviance instead of the change in the values of coefficient estimates. Convergence is declared when the relative change in the deviance is less than #. More-accurate coefficient estimates are typically achieved by not specifying this option and instead using the default `tolerance(1e-7)` criterion or specifying a smaller value for `tolerance(#)`.

The following option is available with `elasticnet` but is not shown in the dialog box:

`penaltywt(`*matname*`)` is a programmer's option for specifying a vector of weights for the coefficients in the penalty term. The contribution of each coefficient to the lasso penalty term is multiplied by its corresponding weight. Weights must be nonnegative. By default, each coefficient's penalty weight is 1.

Remarks and examples

Elastic net, originally proposed by Zou and Hastie (2005), extends lasso to have a penalty term that is a mixture of the absolute-value penalty used by lasso and the squared penalty used by ridge regression. Coefficient estimates from elastic net are more robust to the presence of highly correlated covariates than are lasso solutions.

For the linear model, the penalized objective function for elastic net is

$$Q = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \beta_0 - \mathbf{x}_i\boldsymbol{\beta}')^2 + \lambda\sum_{j=1}^{p}\left(\frac{1-\alpha}{2}\,\beta_j^2 + \alpha\,|\beta_j|\right)$$

where $\boldsymbol{\beta}$ is the p-dimensional vector of coefficients on covariates $\mathbf{x}$. The estimated $\boldsymbol{\beta}$ are those that minimize Q for given values of α and λ.

As with lasso, p can be greater than the sample size N. When $\alpha = 1$, elastic net reduces to lasso. When $\alpha = 0$, elastic net reduces to ridge regression.

When $\alpha > 0$, elastic net, like lasso, produces sparse solutions in which many of the coefficient estimates are exactly zero. When $\alpha = 0$, that is, ridge regression, all coefficients are nonzero, although typically many are small.

Ridge regression has long been used as a method to keep highly collinear variables in a regression model used for prediction. The ordinary least-squares (OLS) estimator becomes increasingly unstable as the correlation among the covariates grows. OLS produces wild coefficient estimates on highly correlated covariates that cancel each other out in terms of fit. The ridge regression penalty removes this instability and produces point estimates that can be used for prediction in this case.

None of the ridge regression estimates are exactly zero because the squared penalty induces a smooth tradeoff around 0 instead of the kinked-corner tradeoff induced by lasso. By mixing the two penalties, elastic net retains the sparse-solution property of lasso, but it is less variable than the lasso in the presence of highly collinear variables. The coefficient paths of elastic-net solutions are also smoother over λ than are lasso solutions because of the added ridge-regression component.

To fit a model with `elasticnet`, you specify a set of candidate α's and a grid of λ values. CV is performed on the combined set of (α, λ) values, and the (α^*, λ^*) pair that minimizes the value of the CV function is selected.

This procedure follows the convention of Hastie, Tibshirani, and Wainwright (2015), which is to specify a few values for α and a finer grid for λ. The idea is that only a few points in the space between ridge regression and lasso are worth reviewing, but a finer grid over λ is needed to trace out the paths of which coefficients are not zero.

The default candidate values of α are 0.5, 0.75, and 1. Typically, you would use the default first and then set α using the `alpha(`*numlist*`)` option to get lower and upper bounds on α^*. Models for small, nonzero values of α take more time to estimate than $\alpha = 0$ and larger values of α. This is because the algorithm for fitting a model that is mostly ridge regression with a little lasso mixed in is inherently inefficient. Pure ridge or mostly lasso models are faster.

The λ grid is set automatically, and the default settings are typically sufficient to determine λ^*. The default grid can be changed using the `grid()` option. See [LASSO] **lasso fitting** for a detailed description of the CV selection process and how to set options to control it.

▷ Example 1: Elastic net and data that are not highly correlated

We will fit an elastic-net model using the example dataset from [LASSO] **lasso examples**. It has stored variable lists created by `vl`. See [D] **vl** for a complete description of the `vl` system and how to use it to manage large variable lists.

After we load the dataset, we type `vl rebuild` to make the saved variable lists active again.

```
. use https://www.stata-press.com/data/r19/fakesurvey_vl
(Fictitious survey data with vl)

. vl rebuild
Rebuilding vl macros ...

-------------------------------------------------------------------------------
                    |                     Macro's contents
                    |----------------------------------------------------------
Macro               |  # Vars   Description
--------------------+----------------------------------------------------------
System              |
  $vldummy          |      98   0/1 variables
  $vlcategorical    |      16   categorical variables
  $vlcontinuous     |      29   continuous variables
  $vluncertain      |      16   perhaps continuous, perhaps categorical variables
  $vlother          |      12   all missing or constant variables
User                |
  $demographics     |       4   variables
  $factors          |     110   variables
  $idemographics    |           factor-variable list
  $ifactors         |           factor-variable list
-------------------------------------------------------------------------------
```

We have four user-defined variable lists, `demographics`, `factors`, `idemographics`, and `ifactors`. The variable lists `idemographics` and `ifactors` contain factor-variable versions of the categorical variables in `demographics` and `factors`. That is, a variable `q3` in `demographics` is `i.q3` in `idemographics`. See [LASSO] **lasso examples** to see how we created these variable lists.

We are going to use `idemographics` and `ifactors` along with the system-defined variable list `vlcontinuous` as arguments to `elasticnet`. Together they contain the potential variables we want to specify. Variable lists are actually global macros, and when we use them as arguments in commands, we put a `$` in front of them.

We also set the random-number seed using the `rseed()` option so we can reproduce our results.

```
. elasticnet linear q104 $idemographics $ifactors $vlcontinuous, rseed(1234)

alpha 1 of 3: alpha = 1

10-fold cross-validation with 109 lambdas ...
Grid value 1:     lambda = 1.818102   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 18.34476
  (output omitted)
Grid value 37:    lambda = .0737359   no. of nonzero coef. =  80
Folds: 1...5....10   CVF = 11.92887
... cross-validation complete ... minimum found

alpha 2 of 3: alpha = 0.75

10-fold cross-validation with 109 lambdas ...
Grid value 1:     lambda = 1.818102   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 18.34476
  (output omitted)
Grid value 34:    lambda = .0974746   no. of nonzero coef. = 126
Folds: 1...5....10   CVF = 11.95437
... cross-validation complete ... minimum found
```

```
alpha 3 of 3: alpha = 0.5

10-fold cross-validation with 109 lambdas ...
Grid value 1:     lambda = 1.818102   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 18.33643
  (output omitted)
Grid value 31:    lambda = .1288556   no. of nonzero coef. = 139
Folds: 1...5....10   CVF =  12.0549
... cross-validation complete ... minimum found

Elastic net linear model                         No. of obs         =        914
                                                 No. of covariates =        277
Selection: Cross-validation                      No. of CV folds    =         10

───────────────┬──────────────────────────────────────────────────────────────
               │                                   No. of   Out-of-      CV mean
               │                                  nonzero    sample   prediction
alpha       ID │     Description      lambda        coef.  R-squared        error
───────────────┼──────────────────────────────────────────────────────────────
1.000          │
             1 │    first lambda    1.818102            0    -0.0016     18.34476
            32 │   lambda before    .1174085           58     0.3543     11.82553
          * 33 │ selected lambda    .1069782           64     0.3547     11.81814
            34 │    lambda after    .0974746           66     0.3545      11.8222
            37 │     last lambda    .0737359           80     0.3487     11.92887
───────────────┼──────────────────────────────────────────────────────────────
0.750          │
            38 │    first lambda    1.818102            0    -0.0016     18.34476
            71 │     last lambda    .0974746          126     0.3473     11.95437
───────────────┼──────────────────────────────────────────────────────────────
0.500          │
            72 │    first lambda    1.818102            0    -0.0012     18.33643
           102 │     last lambda    .1288556          139     0.3418      12.0549
───────────────┴──────────────────────────────────────────────────────────────
* alpha and lambda selected by cross-validation.
```

CV selected $\alpha^* = 1$, that is, the results from an ordinary lasso.

All models we fit using elastic net on these data selected $\alpha^* = 1$. The data are not correlated enough to need elastic net.

◁

▷ Example 2: Elastic net and data that are highly correlated

The dataset in example 1, `fakesurvey_vl`, contained data we created in a simulation. We did our simulation again setting the correlation parameters to much higher values, up to $\rho = 0.95$, and we created two groups of highly correlated variables, with correlations between variables from different groups much lower. We saved these data in a new dataset named `fakesurvey2_vl`. Elastic net was proposed not just for highly correlated variables but especially for groups of highly correlated variables.

We load the new dataset and run `vl rebuild`.

```
. use https://www.stata-press.com/data/r19/fakesurvey2_vl, clear
(Fictitious survey data with vl)

. vl rebuild
Rebuilding vl macros ...
  (output omitted)
```

In anticipation of elastic net showing interesting results this time, we randomly split our data into two samples of equal sizes. One we will fit models on, and the other we will use to test their predictions. We use splitsample to generate a variable indicating the samples.

```
. set seed 1234

. splitsample, generate(sample) nsplit(2)

. label define svalues 1 "Training" 2 "Testing"

. label values sample svalues
```

We fit an elastic-net model using the default α's.

```
. elasticnet linear q104 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234)

alpha 1 of 3: alpha = 1
  (output omitted)
10-fold cross-validation with 109 lambdas ...
Grid value 1:     lambda = 6.323778   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 26.82324
  (output omitted)
Grid value 42:    lambda =  .161071   no. of nonzero coef. =  29
Folds: 1...5....10   CVF = 15.12964
... cross-validation complete ... minimum found

alpha 2 of 3: alpha = 0.75
  (output omitted)
10-fold cross-validation with 109 lambdas ...
Grid value 1:     lambda = 6.323778   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 26.82324
  (output omitted)
Grid value 40:    lambda = .1940106   no. of nonzero coef. =  52
Folds: 1...5....10   CVF = 15.07523
... cross-validation complete ... minimum found

alpha 3 of 3: alpha = 0.5
  (output omitted)
10-fold cross-validation with 109 lambdas ...
Grid value 1:     lambda = 6.323778   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 26.78722
  (output omitted)
Grid value 46:    lambda =   .11102   no. of nonzero coef. = 115
Folds: 1...5....10   CVF = 14.90808
... cross-validation complete ... minimum found
```

```
Elastic net linear model                         No. of obs        =        449
                                                 No. of covariates =        275
Selection: Cross-validation                      No. of CV folds   =         10

------------------------------------------------------------------------------
              |                                   No. of    Out-of-     CV mean
              |                                  nonzero     sample  prediction
alpha      ID |     Description      lambda        coef.  R-squared       error
--------------+---------------------------------------------------------------
1.000         |
            1 |    first lambda    6.323778            0    -0.0036    26.82324
           42 |     last lambda     .161071           29     0.4339    15.12964
--------------+---------------------------------------------------------------
0.750         |
           43 |    first lambda    6.323778            0    -0.0036    26.82324
           82 |     last lambda    .1940106           52     0.4360    15.07523
--------------+---------------------------------------------------------------
0.500         |
           83 |    first lambda    6.323778            0    -0.0022    26.78722
          124 |   lambda before     .161071           87     0.4473    14.77189
        * 125 | selected lambda    .1467619           92     0.4476    14.76569
          126 |    lambda after     .133724           96     0.4468    14.78648
          128 |     last lambda      .11102          115     0.4422    14.90808
------------------------------------------------------------------------------
* alpha and lambda selected by cross-validation.

. estimates store elasticnet
```

Wonderful! It selected $\alpha^* = 0.5$. We should not stop here, however. There may be smaller values of α that give lower minimums of the CV function. If the number of observations and number of potential variables are not too large, you could specify the option `alpha(0(0.1)1)` the first time you run `elasticnet`. However, if we did this, the command would take much longer to run than the default. It will be especially slow for $\alpha = 0.1$ as we mentioned earlier.

```
. elasticnet linear q104 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234) alpha(0.1 0.2 0.3)

alpha 1 of 3: alpha = .3
  (output omitted)
10-fold cross-validation with 113 lambdas ...
Grid value 1:     lambda = 31.61889   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 26.82324
  (output omitted)
Grid value 59:    lambda =  .160193   no. of nonzero coef. = 122
Folds: 1...5....10   CVF = 14.84229
... cross-validation complete ... minimum found

alpha 2 of 3: alpha = .2
  (output omitted)
10-fold cross-validation with 113 lambdas ...
Grid value 1:     lambda = 31.61889   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 26.82324
  (output omitted)
```

```
Grid value 56:    lambda = .2117657   no. of nonzero coef. = 137
Folds: 1...5....10   CVF = 14.81594
... cross-validation complete ... minimum found
alpha 3 of 3: alpha = .1
  (output omitted)
10-fold cross-validation with 113 lambdas ...
Grid value 1:     lambda = 31.61889   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 26.81813
  (output omitted)
Grid value 51:    lambda = .3371909   no. of nonzero coef. = 162
Folds: 1...5....10   CVF = 14.81783
... cross-validation complete ... minimum found
Elastic net linear model                          No. of obs        =        449
                                                  No. of covariates =        275
Selection: Cross-validation                       No. of CV folds   =         10
```

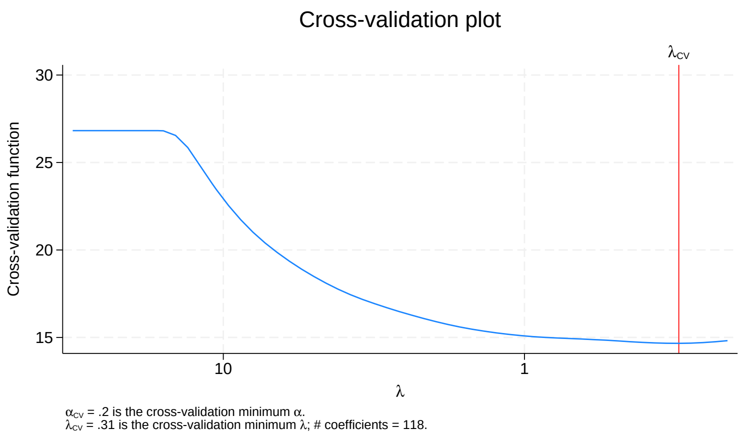
alpha	ID	Description	lambda	No. of nonzero coef.	Out-of-sample R-squared	CV mean prediction error
0.300						
	1	first lambda	31.61889	0	-0.0036	26.82324
	59	last lambda	.160193	122	0.4447	14.84229
0.200						
	60	first lambda	31.61889	0	-0.0036	26.82324
	110	lambda before	.3371909	108	0.4512	14.66875
	* 111	selected lambda	.3072358	118	0.4514	14.66358
	112	lambda after	.2799418	125	0.4509	14.67566
	115	last lambda	.2117657	137	0.4457	14.81594
0.100						
	116	first lambda	31.61889	0	-0.0034	26.81813
	166	last lambda	.3371909	162	0.4456	14.81783

```
* alpha and lambda selected by cross-validation.
. estimates store elasticnet
```

The selected α^* is 0.2. This value is better, according to CV, than $\alpha = 0.1$ or $\alpha = 0.3$.

We can plot the CV function for the selected $\alpha^* = 0.2$.

```
. cvplot
```



The CV function looks quite flat around the selected λ^*. We could assess alternative λ (and alternative α) using `lassoknots`. We run `lassoknots` with options requesting the number of nonzero coefficients be shown (`nonzero`), along with the CV function (`cvmpe`) and estimates of the out-of-sample R^2 (`osr2`).

```
. lassoknots, display(nonzero cvmpe osr2)
```

```
─────────────────────────────────────────────────────────────────
                                   No. of    CV mean     Out-of-
                                  nonzero      pred.      sample
alpha        ID │      lambda      coef.      error   R-squared
────────────────┼────────────────────────────────────────────────
0.300           │
             15 │    9.603319          4   26.42296      0.0114
  (output omitted)
             54 │    .2550726         92   14.67746      0.4509
             55 │    .2324126         98   14.66803      0.4512
             56 │    .2117657        105   14.67652      0.4509
  (output omitted)
             59 │     .160193        122   14.84229      0.4447
────────────────┼────────────────────────────────────────────────
0.200           │
             69 │    14.40498          4   26.54791      0.0067
  (output omitted)
            110 │    .3371909        108   14.66875      0.4512
          * 111 │    .3072358        118   14.66358      0.4514
            112 │    .2799418        125   14.67566      0.4509
  (output omitted)
            115 │    .2117657        137   14.81594      0.4457
```

```
──────────────┬─────────────────────────────────────────
0.100         │
          117 │ 28.80996          4   26.67947       0.0018
  (output omitted)
          161 │ .5369033        143   14.76586       0.4476
          162 │ .4892063        148   14.75827       0.4478
          162 │ .4892063        148   14.75827       0.4478
          163 │ .4457466        152   14.76197       0.4477
  (output omitted)
          166 │ .3371909        162   14.81783       0.4456
──────────────┴─────────────────────────────────────────

* alpha and lambda selected by cross-validation.
```

When we examine the output from `lassoknots`, we see that the CV function appears rather flat along λ from the minimum and also across α.

◁

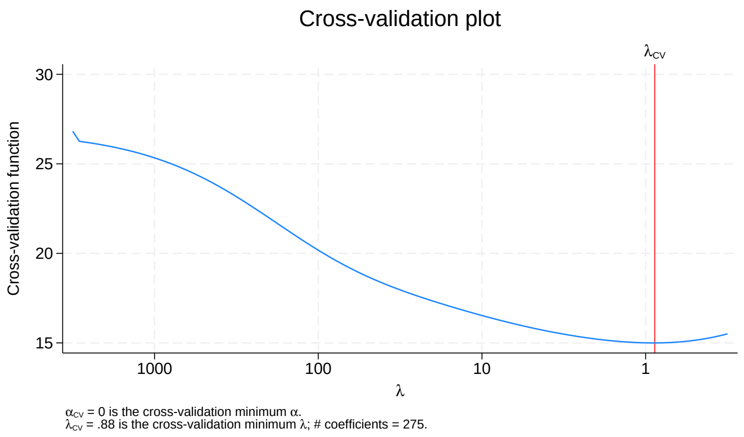
▷ Example 3: Ridge regression

Let's continue with the previous example and fit a ridge regression. We do this by specifying `alpha(0)`.

```
. elasticnet linear q104 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234) alpha(0)
  (output omitted)
Evaluating up to 100 lambdas in grid ...
Grid value 1:     lambda = 3.16e+08   no. of nonzero coef. = 275
Grid value 2:     lambda = 2880.996   no. of nonzero coef. = 275
  (output omitted)
Grid value 99:    lambda = .3470169   no. of nonzero coef. = 275
Grid value 100:   lambda = .3161889   no. of nonzero coef. = 275
10-fold cross-validation with 100 lambdas ...
Fold  1 of 10:  10....20....30....40....50....60....70....80....90....100
  (output omitted)
Fold 10 of 10:  10....20....30....40....50....60....70....80....90....100
... cross-validation complete
Elastic net linear model                         No. of obs         =        449
                                                 No. of covariates =        275
Selection: Cross-validation                      No. of CV folds    =         10

───────────────┬──────────────────────────────────────────────────────────────
               │                               No. of      Out-of-      CV mean
               │                              nonzero       sample   prediction
alpha       ID │     Description      lambda    coef.    R-squared        error
───────────────┼──────────────────────────────────────────────────────────────
0.000          │
             1 │    first lambda    3161.889      275      -0.0036     26.82323
            88 │   lambda before    .9655953      275       0.4387     15.00168
          * 89 │ selected lambda    .8798144      275       0.4388     14.99956
            90 │    lambda after    .8016542      275       0.4386     15.00425
           100 │     last lambda    .3161889      275       0.4198     15.50644
───────────────┴──────────────────────────────────────────────────────────────

* alpha and lambda selected by cross-validation.
. estimates store ridge
```

In this implementation, ridge regression selects λ^* using CV. We can plot the CV function.

```
. cvplot
```



◁

▷ Example 4: Comparing elastic net, ridge regression, and lasso

We fit elastic net and ridge on half of the sample in the previous examples so we could evaluate the prediction on the other half of the sample.

Let's continue with the data from example 2 and example 3 and fit a lasso.

```
. lasso linear q104 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234)
note: 1.q14 omitted because of collinearity with another variable.
note: 1.q136 omitted because of collinearity with another variable.
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = 3.161889   no. of nonzero coef. =   0
```

(output omitted)

```
Grid value 33:    lambda =  .161071   no. of nonzero coef. =  29
Folds: 1...5....10   CVF = 15.12964
... cross-validation complete ... minimum found

Lasso linear model                          No. of obs        =        449
                                            No. of covariates =        275
Selection: Cross-validation                 No. of CV folds   =         10

--------------------------------------------------------------------------
         |                                No. of      Out-of-      CV mean
         |                               nonzero       sample   prediction
      ID |     Description      lambda     coef.    R-squared        error
---------+----------------------------------------------------------------
       1 |    first lambda    3.161889         0       0.0020     26.67513
      28 |   lambda before    .2564706        18       0.4348     15.10566
    * 29 | selected lambda    .2336864        21       0.4358     15.07917
      30 |    lambda after    .2129264        21       0.4355     15.08812
      33 |     last lambda     .161071        29       0.4339     15.12964
--------------------------------------------------------------------------
* lambda selected by cross-validation.

. estimates store lasso
```

We stored the results of the earlier elastic net and ridge in memory using `estimates store`. We did the same for the lasso results. Now we can compare out-of-sample prediction using `lassogof`.

```
. lassogof elasticnet ridge lasso, over(sample)
Penalized coefficients
─────────────────────────────────────────────────────────────────
Name                  sample │         MSE    R-squared        Obs
─────────────────────────────┼───────────────────────────────────
elasticnet                   │
                    Training │    11.70471       0.5520        480
                     Testing │    14.60949       0.4967        501
─────────────────────────────┼───────────────────────────────────
ridge                        │
                    Training │    11.82482       0.5576        449
                     Testing │    14.88123       0.4809        476
─────────────────────────────┼───────────────────────────────────
lasso                        │
                    Training │    13.41709       0.4823        506
                     Testing │    14.91674       0.4867        513
─────────────────────────────────────────────────────────────────
```

Elastic net did better out of sample based on the mean squared error and R^2 than ridge and lasso.

Note that the numbers of observations for both the training and testing samples were slightly different for each of the models. `splitsample` split the sample exactly in half with 529 observations in each half sample. The sample sizes across the models differ because the different models contain different sets of selected variables; hence, the pattern of missing values is different. If you want to make the half samples exactly equal after missing values are dropped, an optional varlist containing the dependent variable and all the potential variables can be used with `splitsample` to omit any missing values in these variables. See [D] **splitsample**.

Before we conclude that elastic net won out over ridge and lasso, we must point out that we were not fair to lasso. Theory states that for the lasso linear model, postselection coefficients provide slightly better predictions. See `predict` in [LASSO] **lasso postestimation**.

We run `lassogof` again for the lasso results, this time specifying that postselection coefficients be used.

```
. lassogof lasso, over(sample) postselection
Postselection coefficients
─────────────────────────────────────────────────────────────────
Name                  sample │         MSE    R-squared        Obs
─────────────────────────────┼───────────────────────────────────
lasso                        │
                    Training │    13.14487       0.4928        506
                     Testing │    14.62903       0.4966        513
─────────────────────────────────────────────────────────────────
```

We declare a tie with elastic net!

Postselection coefficients should not be used with `elasticnet` and, in particular, with ridge regression. Ridge works by shrinking the coefficient estimates, and these are the estimates that should be used for prediction. Because postselection coefficients are OLS regression coefficients for the selected coefficients and because ridge always selects all variables, postselection coefficients after ridge are OLS regression coefficients for all potential variables, which clearly we do not want to use for prediction.

◁

Stored results

elasticnet stores the following in e():

Scalars

e(N)	number of observations
e(N_clust)	number of clusters
e(k_allvars)	number of potential variables
e(k_nonzero_sel)	number of nonzero coefficients for selected model
e(k_nonzero_cv)	number of nonzero coefficients at CV mean function minimum
e(k_nonzero_serule)	number of nonzero coefficients for one-standard-error rule
e(k_nonzero_min)	minimum number of nonzero coefficients among estimated λ's
e(k_nonzero_max)	maximum number of nonzero coefficients among estimated λ's
e(k_nonzero_bic)	number of nonzero coefficients at BIC function minimum
e(alpha_sel)	value of selected α^*
e(alpha_cv)	value of α at CV mean function minimum
e(lambda_sel)	value of selected λ^*
e(lambda_gmin)	value of λ at grid minimum
e(lambda_gmax)	value of λ at grid maximum
e(lambda_last)	value of last λ computed
e(lambda_cv)	value of λ at CV mean function minimum
e(lambda_serule)	value of λ for one-standard-error rule
e(lambda_bic)	value of λ at BIC function minimum
e(ID_sel)	ID of selected λ^*
e(ID_cv)	ID of λ at CV mean function minimum
e(ID_serule)	ID of λ for one-standard-error rule
e(ID_bic)	ID of λ at BIC function minimum
e(cvm_min)	minimum CV mean function value
e(cvm_serule)	CV mean function value at one-standard-error rule
e(devratio_min)	minimum deviance ratio
e(devratio_max)	maximum deviance ratio
e(L1_min)	minimum value of ℓ_1-norm of penalized unstandardized coefficients
e(L1_max)	maximum value of ℓ_1-norm of penalized unstandardized coefficients
e(L2_min)	minimum value of ℓ_2-norm of penalized unstandardized coefficients
e(L2_max)	maximum value of ℓ_2-norm of penalized unstandardized coefficients
e(ll_sel)	log-likelihood value of selected model
e(n_lambda)	number of λ's
e(n_fold)	number of CV folds
e(stop)	stopping rule tolerance

Macros

e(cmd)	elasticnet
e(cmdline)	command as typed
e(depvar)	name of dependent variable
e(allvars)	names of all potential variables
e(allvars_sel)	names of all selected variables
e(alwaysvars)	names of always-included variables
e(othervars_sel)	names of other selected variables
e(post_sel_vars)	all variables needed for postelastic net
e(clustvar)	name of cluster variable
e(lasso_selection)	selection method
e(sel_criterion)	criterion used to select λ^*
e(crossgrid)	type of two-dimensional grid
e(model)	linear, logit, probit, poisson, or cox
e(title)	title in estimation output
e(rngstate)	random-number state used
e(properties)	b
e(predict)	program used to implement predict
e(marginsnotok)	predictions disallowed by margins

Matrices

`e(b)`	penalized unstandardized coefficient vector
`e(b_standardized)`	penalized standardized coefficient vector
`e(b_postselection)`	postselection coefficient vector

Functions

`e(sample)`	marks estimation sample

In addition to the above, the following is stored in `r()`:

Matrices

`r(table)`	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

The methods and formulas for elastic net are given in *Methods and formulas* in [LASSO] **lasso**. Here we provide the methods and formulas for ridge regression, which is a special case of elastic net.

Unlike lasso and elastic net, ridge regression has a differentiable objective function, and there is a closed-form solution to the problem of minimizing the objective function. The solutions for ridge regression with nonlinear models are obtained by iteratively reweighted least squares.

The estimates of a generalized linear model (GLM) ridge regression model are obtained by minimizing

$$Q_L = \sum_{i=1}^{N} \widetilde{w}_i f(y_i, \beta_0 + \mathbf{x}_i\boldsymbol{\beta}') + \frac{\lambda}{2}\sum_{j=1}^{p}\kappa_j\beta_j^2$$

where N is the number of observations; $\widetilde{w}_i$ is the normalized observation-level weight; $f(\cdot)$ is the likelihood contribution for the `regress`, `logit`, `probit`, or `poisson` model; β_0 is the intercept; $\mathbf{x}_i$ is the $1 \times p$ vector of covariates; $\boldsymbol{\beta}$ is the $1 \times p$ vector of coefficients; λ is the ridge penalty parameter, which must be greater than or equal to 0; and κ_j are coefficient-level weights (which by default are all 1).

The normalized weights $\widetilde{w}_i$ sum to 1. That is,

$$\widetilde{w}_i = \frac{w_i}{\sum_{i=1}^{N} w_i}$$

where w_i is the original observation-level weight. If weights are not specified with `elasticnet`, $w_i = 1$ and $\widetilde{w}_i = 1/N$.

The penalized objective function of the ridge regression for the `cox` model is

$$Q_L = -\sum_{j=1}^{N_f}\sum_{i\in D_j}\widetilde{w}_i\left[\mathbf{x}_i\boldsymbol{\beta}' - \log\left\{\sum_{\ell\in R_j}\widetilde{w}_\ell\exp(\mathbf{x}_\ell\boldsymbol{\beta}')\right\}\right] + \frac{\lambda}{2}\sum_{j=1}^{p}\kappa_j\beta_j^2$$

where j indexes the ordered failure times $t_{(j)}$, $j = 1, \dots, N_f$; D_j is the set of observations that fail at $t_{(j)}$; and R_j is the set of observations k that are at risk at time $t_{(j)}$ (that is, all k such that $t_{0k} < t_{(j)} \le t_k$, and t_{0k} is the entry time for the kth observation).

When the model is `linear`,

$$f(y_i, \beta_0 + \mathbf{x}_i\boldsymbol{\beta}') = \frac{1}{2}(y_i - \beta_0 - \mathbf{x}_i\boldsymbol{\beta}')^2$$

When the model is `logit`,

$$f(y_i, \beta_0 + \mathbf{x}_i\boldsymbol{\beta}') = -y_i(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}') + \ln\{1 + \exp(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')\}$$

When the model is `probit`,

$$f(y_i, \beta_0 + \mathbf{x}_i\boldsymbol{\beta}') = -y_i \ln\left\{\Phi(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')\right\} - (1 - y_i)\ln\left\{1 - \Phi(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')\right\}$$

When the model is `poisson`,

$$f(y_i, \beta_0 + \mathbf{x}_i\boldsymbol{\beta}') = -y_i(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}') + \exp(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')$$

For the `linear` model, the point estimates are given by

$$(\widehat{\beta}_0, \widehat{\boldsymbol{\beta}})' = \left(\sum_{i=1}^{N} \widetilde{w}_i \tilde{\mathbf{x}}_i' \tilde{\mathbf{x}}_i + \lambda\tilde{\mathbf{I}}\right)^{-1} \sum_{i=1}^{N} \widetilde{w}_i y_i \tilde{\mathbf{x}}_i'$$

where $\tilde{\mathbf{x}}_i = (1, \mathbf{x}_i)$ and $\tilde{\mathbf{I}}$ is a diagonal matrix with the coefficient-level weights $0, \kappa_1, \ldots, \kappa_p$ on the diagonal.

For the nonlinear models, the optimization problem is solved using iteratively reweighted least squares. See Segerstedt (1992) and Nyquist (1991) for details of the iteratively reweighted least-squares algorithm for the GLM ridge-regression estimator.

References

Hastie, T. J., R. J. Tibshirani, and M. Wainwright. 2015. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Boca Raton, FL: CRC Press. https://doi.org/10.1201/b18401.

Nyquist, H. 1991. Restricted estimation of generalized linear models. *Journal of the Royal Statistical Society,* C ser., 40: 133–141. https://doi.org/10.2307/2347912.

Segerstedt, B. 1992. On ordinary ridge regression in generalized linear models. *Communications in Statistics—Theory and Methods* 21: 2227–2246. https://doi.org/10.1080/03610929208830909.

Zou, H., and T. J. Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society,* B ser., 67: 301–320. https://doi.org/10.1111/j.1467-9868.2005.00503.x.

Also see

[LASSO] **lasso postestimation** — Postestimation tools for lasso for prediction

[LASSO] **lasso** — Lasso for prediction and model selection

[LASSO] **Lasso intro** — Introduction to lasso

[LASSO] **sqrtlasso** — Square-root lasso for prediction and model selection

[R] **logit** — Logistic regression, reporting coefficients

[R] **poisson** — Poisson regression

[R] **probit** — Probit regression

[R] **regress** — Linear regression

[ST] **stset** — Declare data to be survival-time data

[U] 20 Estimation and postestimation commands

estimates store — Saving and restoring estimates in memory and on disk

Description Remarks and examples Also see

Description

`estimates store` *name* stores the current (active) estimation results under the name *name*.

`estimates restore` *name* loads the results stored under *name* into the current (active) estimation results.

`estimates save` *filename* saves the current (active) estimation results in *filename*.

`estimates use` *filename* loads the results saved in *filename* into the current (active) estimation results.

The `estimates` commands after the lasso commands work the same as they do after other estimation commands. There is only one difference. `estimates save` *filename* saves two files, not just one. *filename*`.ster` and *filename*`.stxer` are saved. See [R] **estimates** for details.

Remarks and examples

Remarks are presented under the following headings:

Overview

If you are not familiar with `estimates store` and `restore`, see [R] **estimates store**. You will likely want to use `estimates store` to compare results from multiple lassos.

If you are not familiar with `estimates save` and `use`, see [R] **estimates save**. Lassos fit with many potential variables can take considerable time to run. `xpo` commands, especially when the `resample` option is specified, can also have lengthy computation times. You will likely want to save your estimation results to a file.

When you use `estimates save`, you will see

```
. estimates save mygreatlasso
file mygreatlasso.ster saved
extended file mygreatlasso.stxer saved
```

Two files are saved. Keep these files together in the same folder (directory). `estimates use` needs both of them to load the results back into the current estimation results.

Postestimation commands that work only with current results

The following postestimation commands work only with current (active) estimation results: `bicplot`, `coefpath`, `cvplot`, `lassoknots`, and `lassoselect`.

The following postestimation commands work with current or stored estimation results: `lassocoef`, `lassogof`, and `lassoinfo`.

For the commands that work only with current results, this means that if you

```
. estimates store mylasso1
```

and then run another estimation command, you must

```
        . estimates restore lasso1
```

before you can use bicplot, coefpath, cvplot, lassoknots, or lassoselect again.

Postestimation commands that work with current results

lassocoef and lassogof are intended for use with multiple estimation results. You will often be typing commands such as

```
        . lassgof mylasso1 mylasso2 mylasso3, over(sample)
```

where mylasso1, mylasso2, and mylasso3 are names of stored estimation results. See [LASSO] **lassogof** for examples.

lassocoef has a more complex syntax because it will work with lasso, sqrtlasso, and elasticnet, and also with the ds, po, and xpo commands or a mixture of them. You can type something like

```
        . lassocoef mylasso1 (mydsregress, for(y)) (mydsregress, for(x))
```

where mylasso1 and mydsregress are names of stored estimation results, with mylasso1 a lasso result and mydsregress a dsregress result. See [LASSO] **lassocoef** for examples. lassoinfo is designed to tell you the available names (typically variable names) that can be specified with for().

lassoselect creates new estimation results

When you run one of the lasso commands, such as

```
        . lasso ...
```

and then use lassoselect to change the selected λ^* like so

```
        . lassoselect lambda = 0.245
```

lassoselect creates a new estimation result and makes it current. It is almost the same as running another estimation command and wiping out the old estimation results. We say “almost” because it is easy to change λ^* back to what it was originally.

A better workflow when using lassoselect is the following:

```
        . lasso ...
        . estimates store mylasso1
        . lassoselect lambda = 0.245
        . estimates store mylasso1sel
        . lassogof mylasso1 mylasso1sel, over(sample)
```

See [LASSO] **lassoselect**.

Also see

[R] **estimates save** — Save and use estimation results

[R] **estimates store** — Store and restore estimation results

Inference examples — Examples and workflow for inference

Description   Remarks and examples   References   Also see

Description

Lasso for inference comprises 11 related estimation commands and several postestimation commands for performing inference about a true model. Fitting and interpreting inferential lasso models is demonstrated via examples.

Remarks and examples

Remarks are presented under the following major headings:

1 Overview

1.1 How to read the example entries

All the examples demonstrate something about the inferential lasso models, so we obviously think you should read this entire section. That said, there are a lot of pages, so here are some other options.

Everyone should read *1.3 Review of concepts*, *2.1 Overview of inferential estimation methods*, and *2.2 Fitting via cross-fit partialing out (xpo) using plugin*. What you read there is essential to using and understanding all the inferential models. We are pretty sure you will also want to read *2.3 Fitting via cross-fit partialing out (xpo) using cross-validation*, *2.4 Fitting via double selection (ds) using cross-validation*, and *2.5 Fitting via the other 22 methods*.

We use the variable-management tool `vl` to manage the variable lists used in all the examples, and most of the examples use a common dataset. We introduce both in *1.4 The primary dataset*. We say enough in sections 2.1 and 2.2 that you will not be lost if you do not read section 1.4. But you will better understand the dataset—and how we are manipulating it—if you read section 1.4.

If you are only interested in logit models for binary outcomes, then *3 Fitting logit inferential models to binary outcomes. What is different?* is essential reading, but only after reading sections 1.3, 2.1, and 2.2. Similarly, if your sole interest is Poisson models for count outcomes, then read *4 Fitting inferential models to count outcomes. What is different?*, but only after reading sections 1.3, 2.1, and 2.2.

The titles on all other sections are relatively self explanatory. So, if you are not reading all the sections, choose from them based on your interest.

1.2 Detailed outline of the topics

1.3 Review of concepts

We have said a lot about the inferential estimation commands elsewhere in this manual. For a quick overview that describes what you need to know, and just what you need to know, see [LASSO] **Lasso intro**. For a deeper understanding of lasso for inference, read [LASSO] **Lasso inference intro**. We highly recommend reading both of those sections.

The inferential lasso estimators require you to break up your model into two parts: the part about which you need to perform inference and the part about which you do not care. Let's call the first part the "inference part" and the second part the "noninference part".

Often, the inference part is a single variable, perhaps even a single indicator variable, such as "walks at least three miles a week". The inference part could be more complicated than a single variable. It might involve several variables, polynomials, or interactions. But, it will generally be relatively small.

The noninference part can be much larger. What you include there will sometimes reflect an ignorance of how that part relates to your outcome. Often, our theory or intuition involves only a few variables, our variables of interest. We know lots of other things affect our outcome; we just have little or no guidance about which things are important or how they relate to our outcome. We will call the variables in this noninference part controls. What makes lasso for inference special is that you need not understand how those controls affect the outcome.

There are other requirements. We said that the inference part will typically be small. The number of controls that lasso needs to include must also be small with respect to the sample size. See *Solutions that focus on the true model* in [LASSO] **Lasso inference intro**.

1.4 The primary dataset

To demonstrate the inference commands, we will mostly use one dataset—a real-world dataset that includes children's performance on a test of reaction time, levels of nitrogen dioxide (NO_2) pollution, the children's physical and socioeconomic characteristics, and some other environmental factors. The data were collected and analyzed by Sunyer et al. (2017).

Our interest is in how levels of nitrogen dioxide in the classroom affect the children's performance on the test, while adjusting for other factors. We will focus on two outcomes from the Attention Network Test (ANT)—reaction time and omissions. For linear models, we will use hit reaction time—a measure of speed in responding to stimuli. For Poisson models, we will use omissions—the number of times the child failed to respond to a stimulus. For logit models, we will use whether there were any omissions.

We are using an extract of the data and focusing on how to use the software, so let's not get ideas about publishing any of this.

Let's take a quick look at the dataset.

```
. use https://www.stata-press.com/data/r19/breathe
(Nitrogen dioxide and attention)

. describe

Contains data from https://www.stata-press.com/data/r19/breathe.dta
 Observations:         1,089                  Nitrogen dioxide and attention
    Variables:            22                  21 Jun 2024 12:43
                                              (_dta has notes)
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
react           double  %10.0g              * Reaction time (ms)
correct         int     %10.0g              * Number of correct responses
omissions       byte    %10.0g              * Failure to respond to stimulus
no2_class       float   %9.0g                 Classroom NO2 levels (ug/m3)
no2_home        float   %9.0g                 Home NO2 levels (ug/m3)
age             float   %9.0g                 Age (years)
age0            double  %4.1f                 Age started school
sex             byte    %9.0g      sex        Sex
grade           byte    %9.0g      grade      Grade in school
overweight      byte    %32.0g     overwt   * Overweight by WHO/CDC definition
lbweight        byte    %18.0g     lowbw    * Low birthweight
breastfeed      byte    %19.0f     bfeed      Duration of breastfeeding
msmoke          byte    %10.0f     smoke    * Mother smoked during pregnancy
meducation      byte    %17.0g     edu        Mother's education level
feducation      byte    %17.0g     edu        Father's education level
siblings_old    byte    %1.0f                 Number of older siblings in house
siblings_young  byte    %1.0f                 Number of younger siblings in
                                                house
sev_home        float   %9.0g                 Home socio-economic vulnerability
                                                index
green_home      double  %10.0g                Home greenness (NDVI), 300m
                                                buffer
noise_school    float   %9.0g                 School noise levels (dB)
sev_school      float   %9.0g                 School socio-economic
                                                vulnerability index
precip          double  %10.0g                Daily total precipitation
                                            * indicated variables have notes
-------------------------------------------------------------------------------
Sorted by:
```

This is not a large dataset, just 22 variables. Regardless, we are going to use the `vl` tools to create the variable lists we need for our analysis. This may seem like a detour, but `vl` is useful even for small datasets, and it is nearly indispensable if your dataset has hundreds or even tens of thousands of variables.

Our goal is to create two lists of control covariates, for example, independent variables. One list will contain continuous control covariates and the other will contain categorical control covariates. Why not just one list? Because we want the categorical variables to enter our model as indicator variables for each level (distinct value) of the categorical variable. To expand a categorical variable into indicator variables for its levels, we must prefix it with an `i.`, for example, `i.grade`.

Starting with `vl` is easy: we just type `vl set`,

```
. vl set

-------------------------------------------------------------------------------
                  |                      Macro's contents
                  |------------------------------------------------------------
Macro             |  # Vars   Description
------------------+------------------------------------------------------------
System            |
  $vlcategorical  |      10   categorical variables
  $vlcontinuous   |      10   continuous variables
  $vluncertain    |       2   perhaps continuous, perhaps categorical variables
  $vlother        |       0   all missing or constant variables
-------------------------------------------------------------------------------
Notes

      1. Review contents of vlcategorical and vlcontinuous to ensure they are
         correct.  Type vl list vlcategorical and type vl list vlcontinuous.

      2. If there are any variables in vluncertain, you can reallocate them
         to vlcategorical, vlcontinuous, or vlother.  Type
         vl list vluncertain.

      3. Use vl move to move variables among classifications.  For example,
         type vl move (x50 x80) vlcontinuous to move variables x50 and x80 to
         the continuous classification.

      4. vlnames are global macros.  Type the vlname without the leading
         dollar sign ($) when using vl commands.  Example: vlcategorical not
         $vlcategorical.  Type the dollar sign with other Stata commands to
         get a varlist.
```

`vl` has divided our 22 variables into 4 groups and placed those groups into global macros. Do not worry about the technical term global macro. Just know that once the macro `$vlcategorical` has been created, any time you type `$vlcategorical`, you will get the full list of categorical variables.

```
. display "$vlcategorical"
sex grade overweight lbweight breastfeed msmoke meducation feducation
> siblings_old siblings_young
```

That is convenient! `vl` has placed all of our categorical variables into one bin called `$vlcategorical`. Now let's follow the instructions in the notes after `vl` to be sure we like what `vl` did on our behalf.

```
. vl list vlcategorical

        Variable | Macro           Values         Levels
-----------------+--------------------------------------
             sex | $vlcategorical  0 and 1             2
           grade | $vlcategorical  integers >=0        3
      overweight | $vlcategorical  0 and 1             2
        lbweight | $vlcategorical  0 and 1             2
      breastfeed | $vlcategorical  integers >=0        3
          msmoke | $vlcategorical  0 and 1             2
      meducation | $vlcategorical  integers >=0        4
      feducation | $vlcategorical  integers >=0        4
    siblings_old | $vlcategorical  integers >=0        5
  siblings_young | $vlcategorical  integers >=0        5
--------------------------------------------------------
```

Among other things, we see that sex has just two values, 0 and 1; and meducation (mother's education level) has four values that are integers greater than or equal to 0.

Usually with categorical variables, we intend to create indicator variables for each unique value (level) the variable takes on. So we are looking for variables that do not fit that purpose. siblings_old and siblings_young have five values, but even their names make one think they might be counts. Let's look further at siblings_old:

```
. tabulate siblings_old

  Number of
      older
siblings in
      house |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        564       52.17       52.17
          1 |        424       39.22       91.40
          2 |         84        7.77       99.17
          3 |          8        0.74       99.91
          4 |          1        0.09      100.00
------------+-----------------------------------
      Total |      1,081      100.00
```

It does look like a count of siblings. We might want indicators for each count (level), or we might want it to enter our model linearly as a continuous variable. That singleton count of 4 older siblings will have to be dropped whenever we perform cross-validation or cross-fitting because it cannot be in both the estimation and the validation samples. We could recode the values to represent 0, 1, 2, and 3-or-more siblings and keep it a factor variable. After all, lasso is a technique built for handling lots of variables. It is easier for our examples to simply redesignate the two counts of siblings as continuous:

```
. vl move (siblings_old siblings_young) vlcontinuous
note: 2 variables specified and 2 variables moved.

------------------------------
Macro          # Added/Removed
------------------------------
$vlcategorical              -2
$vlcontinuous                2
$vluncertain                 0
$vlother                     0
------------------------------
```

Let's now take a look at all variables designated continuous. We will use `summarize` to get a bit more detail:

```
. summarize $vlcontinuous

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
       react |      1,084    742.4808    145.4446   434.0714    1303.26
   no2_class |      1,089    30.16779    9.895886   7.794096   52.56397
    no2_home |      1,089    54.71832    18.04786   2.076335   118.6568
         age |      1,089     9.08788     .886907    7.45243   11.63313
        age0 |      1,082    3.218022    1.293168          0          9
-------------+---------------------------------------------------------
    sev_home |      1,089    .4196807    .1999143   .0645161   .9677419
  green_home |      1,089    .1980721     .077777   .0184283   .5258679
 noise_school|      1,089    37.96354    4.491651       28.8       51.1
  sev_school |      1,089    .4096389    .2064394   .1290323   .8387097
      precip |      1,089    .5593205      1.2364          0        5.8
-------------+---------------------------------------------------------
siblings_old |      1,081     .573543    .6752252          0          4
siblings_y~g |      1,083     .565097    .6906831          0          6
```

We notice three things. First, `age0` has a min of 0 and a max of 9—both integers. Did `vl set` make a mistake? Let's look carefully:

```
. tabulate age0

Age started |
     school |      Freq.     Percent        Cum.
------------+-----------------------------------
        0.0 |          4        0.37        0.37
        0.3 |          1        0.09        0.46
        0.5 |          1        0.09        0.55
        0.8 |          1        0.09        0.65
        0.9 |          1        0.09        0.74
        1.0 |         33        3.05        3.79
        1.4 |          1        0.09        3.88
        1.5 |          8        0.74        4.62
        1.7 |          1        0.09        4.71
        2.0 |        116       10.72       15.43
        2.5 |          4        0.37       15.80
        2.8 |          3        0.28       16.08
        2.9 |          1        0.09       16.17
        3.0 |        739       68.30       84.47
        4.0 |         35        3.23       87.71
        5.0 |         39        3.60       91.31
        6.0 |         54        4.99       96.30
        7.0 |         25        2.31       98.61
        8.0 |          8        0.74       99.35
        9.0 |          7        0.65      100.00
------------+-----------------------------------
      Total |      1,082      100.00
```

No mistake. There are fractional values. Also, looking back at the results of our `describe`, we see that age 0 is the age at which the students started school. We do want to treat that as continuous.

Second, our dependent variable, `react`, is in the list. It is continuous, and so it belongs there. However, we will need to take care that we do not include it among our control covariates.

Third, our covariate of interest, `no2_class`, is also in the list. As with `react`, we will need to exclude it from the control covariates.

What of those two variables that were in `$vluncertain`?

```
. vl list vluncertain

─────────────────────────────────────────────────────
    Variable │ Macro           Values          Levels
─────────────┼───────────────────────────────────────
     correct │ $vluncertain    integers >=0        41
   omissions │ $vluncertain    integers >=0        27
─────────────────────────────────────────────────────
```

`vl set` knows they are integer, but one has 41 distinct values and the other has 27. `vl` was unwilling to classify them as either continuous or categorical. See [D] **vl** for how to change `vl`'s rules. We said earlier that `omissions` is another outcome variable from the ANT. `correct` is also an outcome variable from the ANT. Both are potential dependent variables, meaning that neither are valid controls. We will leave them where they are.

We were fortunate that `correct` and `omissions` were already left out of `$vlcategorical` and `$vlcontinuous`. Otherwise, it would be our job to ensure they are not included among the controls. `vl` is convenient for classifying variables, but it does not truly understand anything about the meaning of the variables. It is our job to know which variables are actually other outcomes or transformations of the outcomes.

Let's now create our own two global macros. One will have continuous covariates, and we will call that macro `cc`. The other will have categorical covariates, which we will treat as factor covariates, and we will call that macro `fc`.

```
. vl create cc = vlcontinuous - (react no2_class)
note: $cc initialized with 10 variables.

. vl create fc = vlcategorical
note: $fc initialized with 8 variables.
```

`fc` is just a copy of `vlcategorical`. We could just use `vlcategorical`, but it is best to create our own macro in case we want to change it later. When we created `cc`, we removed our dependent variable, `react`, and covariate of interest, `no2_class`. That gives us a list of continuous controls.

Now we have control covariate lists we can use in our inference commands.

No one at StataCorp would ever type everything we just typed interactively. We would open an editor or the Do-file Editor and work there. I suggest you do the same thing. Click on the Do-file Editor button, . Then type in the Editor

```
describe
vl set
vl list vlcategorical
tabulate siblings_old
vl move (siblings_old siblings_young) vlcontinuous
summarize $vlcontinuous
tabulate age0
vl list vluncertain
vl create cc = vlcontinuous - (react no2_class)
vl create fc = vlcategorical
```

Save the file as `no2.do`. Then you can type `do no2` to re-create your control covariate lists.

If you want to exclude the exploratory commands, just type

```
vl set
vl move (siblings_old siblings_young) vlcontinuous
vl create cc = vlcontinuous - (react no2_class)
vl create fc = vlcategorical
```

2 Fitting and interpreting inferential models

2.1 Overview of inferential estimation methods

Considering only the linear models for continuous outcomes and ignoring endogeneity, there are 25 methods to fit any given model. There are three commands—dsregress, poregress, and xporegress. The po and xpo commands allow the option semi, which adjusts how they partial out, making five methods. Within each of these methods, there is an option allowing three ways of selecting the lasso penalty λ—selection(plugin), selection(cv), and selection(adaptive). And, for 10 of these 15 methods, there is an option (sqrtlasso) to specify that the square-root lasso rather than the standard lasso be used to select covariates. Square-root lasso cannot be combined with selection(adaptive).

What you type differs only a little when requesting any of these 25 methods. More importantly, you interpret the coefficient estimates, standard errors, and confidence intervals exactly the same across all 25 methods. Which is to say, you interpret them exactly as you would interpret the estimates from linear regression.

Let's see how to request each of these 25 methods.

Assume that our dependent variable is y. We will include two covariates of interest—d1 and d2. We will specify 100 potential continuous control covariates—x1-x100. And, we have 30 potential factor control variables—f1-f30. The factor variables could be ordered, unordered, or just indicators. We specify them as i.(f1-f30) so that each level of each covariate is included as its own term. So, if f3 has four levels, then it introduces four indicator variables (covariates) into the potential controls. See **[U] 11.4.3 Factor variables**. We could also introduce interactions among the factor variables, among the continuous variables, or both. Do that if you wish.

All these commands will run if you use lassoex.dta.

To make the commands easier to read, we do not specify option rseed() to make reproducible the commands that randomly split the samples repeatedly. If you want the results to be the same each time you run the commands, add rseed(12345) (or whatever number you like).

```
. use https://www.stata-press.com/data/r19/lassoex
```

We can first fit the model using the cross-fit partialing-out method, the partialing-out method, and the double-selection method. In all cases, we are using the default plugin method for choosing the included controls via its choice of the lasso penalty parameter λ.

```
. xporegress y d1 d2, controls(x1-x100 i.(f1-f30))
. poregress  y d1 d2, controls(x1-x100 i.(f1-f30))
. dsregress  y d1 d2, controls(x1-x100 i.(f1-f30))
```

We can fit the same models, but this time using the cross-validation method to choose the lasso penalty parameter λ and thereby to choose the included control covariates.

```
. xporegress y d1 d2, controls(x1-x100 i.(f1-f30)) selection(cv)
. poregress  y d1 d2, controls(x1-x100 i.(f1-f30)) selection(cv)
. dsregress  y d1 d2, controls(x1-x100 i.(f1-f30)) selection(cv)
```

Again, we can fit the same models, but this time using the adaptive method to choose the included control covariates.

```
. xporegress y d1 d2, controls(x1-x100 i.(f1-f30)) selection(adaptive)
. poregress  y d1 d2, controls(x1-x100 i.(f1-f30)) selection(adaptive)
. dsregress  y d1 d2, controls(x1-x100 i.(f1-f30)) selection(adaptive)
```

We can rerun each of the first six methods using the square-root lasso rather than the standard lasso, by adding the option `sqrtlasso`. Here is one example that uses the cross-fit partialing-out method with plugin selection:

```
. xporegress y d1 d2, controls(x1-x100 i.(f1-f30)) sqrtlasso
```

And, we can rerun any of the 10 methods that use commands `poregress` or `xporegress`, including those with `sqrtlasso`, using the `semi` option to specify an alternate form of partialing out. Here is one example:

```
. xporegress y d1 d2, controls(x1-x100 i.(f1-f30)) semi
```

We apologize for the bewildering array of choices. Lasso and machine learning is an active area of research, and you may want the flexibility to choose among these options. That said, if your interest is in your research and not in researching lasso, we feel reasonably comfortable making some suggestions based on the state of the lasso literature at the time this manual was written.

1. Use `xporegress` with no options to fit your model using the cross-fit partialing-out method with λ, and thereby the control covariates, selected using the plugin method.

   The plugin method was designed for variable selection in this inferential framework and has the strongest theoretical justification.

2. If you want to explore the process whereby the control covariates were selected, add option `selection(cv)` to your `xporegress` specification.

   You can then explore the path by which each lasso selected control covariates.

   You are still on firm theoretical footing. Cross-validation meets the requirements of a sufficient variable-selection method.

   Cross-validation has a long history in machine learning. Moreover, what cross-validation is doing and how it chooses the covariates is easy to explain.

3. If you do not want to explore lots of lassos and you want to fit models much more quickly, use commands `dsregress` or `poregress` rather than using `xporegress`.

   `xporegress` fits 10 lassos for the dependent variable and 10 more lassos for each covariate of interest! That is the default; you can request more. Or you can request fewer, but that is not recommended. So, `xporegress` is orders of magnitude slower than `poregress` and `dsregress`. And it has orders of magnitude more lassos to explore. Overwhelming.

   Why then is `xporegress` our first recommendation? It is safer if you think that the process that generated your data has lots of covariates relative to your sample size. Similarly, it is also safer if you want to explore lots of potential controls. The number of potential controls is not as problematic as the number of true covariates because it is the natural log of the potential control that counts. For example, needing 10 additional true covariates is the same as requesting just over 22,000 potential controls. The jargon term for this is sparsity. `xporegress` has a weaker sparsity requirement than do `poregress` and `dsregress`. See *Solutions that focus on the true model* in [LASSO] **Lasso inference intro**.

   Despite this benefit, if your model is weakly identified by the data, `dsregress` can be more stable than either `poregress` or `xporegress`. `dsregress` uses a union of all the selected controls from all the lassos for all of its computations after selection. Both `poregress` and `xporegress` use the results of each lasso separately to perform parts of their computations (specifically, to compute their moments), and then put all that together when solving the moment conditions. This makes poregress and xporegress sensitive to which controls are selected for each lasso. So if you change your specification slightly, dsregress may be more stable. To be clear, we said more stable, not better.

4. We have suggested xporegress without a selection option and xporegress, poregress, and dsregress with option selection(cv). Feel free to try any of the remaining 21 methods. They all meet the requirements of sufficient variable-selection methods, so all can be theoretically justified.

Everything we said above applies to models for binary outcomes fit using xpologit, pologit, and dslogit; and it applies to models for count outcomes fit using xpopoisson, popoisson, and dspoisson.

These suggestions are based on the assumption that you are not concerned that you have violated or are near the method's sparsity bound. See *Solutions that focus on the true model* in [LASSO] **Lasso inference intro** for a discussion of sparsity bounds. Data that fit your model poorly can trigger a sparsity bound sooner than data that fit well. If you are concerned, see some alternate but similar suggestions in [LASSO] **Inference requirements**.

2.2 Fitting via cross-fit partialing out (xpo) using plugin

In the previous section, we recommended using the cross-fit partialing-out estimator xporegress as your first option. We will use that method to fit a model of how levels of nitrogen dioxide (no2_class) in a classroom affect the reaction time (react) of students. We use the dataset described in section 1.4.

```
. use https://www.stata-press.com/data/r19/breathe, clear
(Nitrogen dioxide and attention)
```

We created a do-file in section 1.4 that collects our variables into groups that are convenient for specifying inferential lasso models. If you have it saved, great. We will run the one from the Stata Press website:

```
. do https://www.stata-press.com/data/r19/no2
  (output omitted)
```

Recall that the purpose of the inferential lasso estimators is to estimate the relationship between one, or a few, covariates of interest and a dependent variable, while adjusting for a possibly large set of control variables. And by "large", we mean perhaps many more controls than you have observations.

We now have our list of continuous control variables in global macro $cc and our list of factor-variable control variables in global macro $fc. What does that mean? Anywhere we type $cc, Stata substitutes the list of continuous controls, and anywhere we type $fc, Stata substitutes the list of factor controls. Let's display them:

```
. display "$cc"
no2_home age age0 sev_home green_home noise_school sev_school precip
> siblings_old siblings_young

. display "$fc"
sex grade overweight lbweight breastfeed msmoke meducation feducation
```

That is going to save us a lot of typing.

Now we are ready to fit our model.

```
. xporegress react no2_class, controls($cc i.($fc)) rseed(12345)
Cross-fit fold 1 of 10 ...
Estimating lasso for react using plugin
Estimating lasso for no2_class using plugin
Cross-fit fold 2 of 10 ...
Estimating lasso for react using plugin
Estimating lasso for no2_class using plugin
Cross-fit fold 3 of 10 ...
Estimating lasso for react using plugin
Estimating lasso for no2_class using plugin
Cross-fit fold 4 of 10 ...
Estimating lasso for react using plugin
Estimating lasso for no2_class using plugin
Cross-fit fold 5 of 10 ...
Estimating lasso for react using plugin
Estimating lasso for no2_class using plugin
Cross-fit fold 6 of 10 ...
Estimating lasso for react using plugin
Estimating lasso for no2_class using plugin
Cross-fit fold 7 of 10 ...
Estimating lasso for react using plugin
Estimating lasso for no2_class using plugin
Cross-fit fold 8 of 10 ...
Estimating lasso for react using plugin
Estimating lasso for no2_class using plugin
Cross-fit fold 9 of 10 ...
Estimating lasso for react using plugin
Estimating lasso for no2_class using plugin
Cross-fit fold 10 of 10 ...
Estimating lasso for react using plugin
Estimating lasso for no2_class using plugin
Cross-fit partialing-out            Number of obs                =      1,036
linear model                        Number of controls           =         32
                                    Number of selected controls  =         10
                                    Number of folds in cross-fit =         10
                                    Number of resamples          =          1
                                    Wald chi2(1)                 =      22.87
                                    Prob > chi2                  =     0.0000

------------------------------------------------------------------------------
             |               Robust
       react | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
   no2_class |   2.316063   .4843097     4.78   0.000     1.366834    3.265293
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

The construct `i.($fc)` in `controls()` is factor-variable notation that expands each variable in `$fc` into indicator variables for each distinct value of the variable. We specified `rseed(12345)` to set the seed of the random-number generator so that our results are reproducible. We did this because the cross-fit estimator uses cross-fitting and so divides the sample into random groups. If we do not set the seed, we will get slightly different results each time we run the command. There is nothing special about 12345; choose any number you like. You will get different, but hopefully similar, results for any seed. The same seed will always produce the same results.

Now to the output. That is a long log. `xporegress` is just reporting on its progress as it performs 10 cross-fits and then performs 2 lassos within each group. We see in the header that 1,036 observations were used, that we specified 32 controls, that 10 controls were selected from the 32, and that we did not resample. From the Wald statistic and its p-value, we see that our covariate of interest is highly significant.

We interpret the coefficient estimates just as we would for a standard linear regression. Because this is linear regression, that effect can be interpreted as the population average effect, the effect for any individual, or the effect for any group. What we lose with the inferential lasso estimators is the ability to interpret any other coefficients.

Our point estimate for the effect of nitrogen dioxide on reaction time is 2.3, meaning that we expect reaction time to go up by 2.3 milliseconds for each microgram per cubic meter increase in nitrogen dioxide. This value is statistically different from 0 well beyond the 5% level, in fact, beyond the 0.1% level. Our 95% confidence interval is 1.4 to 3.3.

We also note that `xporegress` estimates robust standard errors, so all the associated statistics are also robust. With `xporegress`, we are robust to nonnormality of the error and to heteroskedasticity.

We can see how stable the lasso selection of controls is by typing `lassoinfo`.

```
. lassoinfo

    Estimate: active
     Command: xporegress

------------------------------------------------------------------------
             |                              No. of selected variables
             |                Selection     ----------------------------
    Variable |      Model        method        min    median        max
-------------+----------------------------------------------------------
   no2_class |     linear        plugin          5         5          5
       react |     linear        plugin          3         5          5
------------------------------------------------------------------------
```

We see that, over the 10 cross-fits, the plugin method selected 5 controls for the lasso on the covariate of interest—`no2_class`. It selected 5 controls every time. For the dependent variable, `react`, the plugin method selected between 3 and 5 controls. Even though these are real data, they look to be easy for the lasso and plugin to handle. There is nothing to interpret in this table, though if some of the lassos are consistently selecting 0 controls, you might want to explore further. See *Solutions that focus on the true model* in [LASSO] **Lasso inference intro** and see [LASSO] **Inference requirements**.

2.3 Fitting via cross-fit partialing out (xpo) using cross-validation

Continuing with the example above, we can use cross-validation to select our controls rather than plugin. Cross-validation is a well-established method in the machine learning literature. Even so, it is known to select more variables than are absolutely necessary. We add `selection(cv)` to our previous `xporegress` command:

```
. xporegress react no2_class, controls($cc i.($fc)) selection(cv) rseed(12345)
  (output omitted)
Cross-fit partialing-out            Number of obs                =      1,036
linear model                        Number of controls           =         32
                                    Number of selected controls  =         26
                                    Number of folds in cross-fit =         10
                                    Number of resamples          =          1
                                    Wald chi2(1)                 =      23.34
                                    Prob > chi2                  =     0.0000

------------------------------------------------------------------------------
             |               Robust
       react | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
   no2_class |   2.348458   .4861133     4.83   0.000     1.395693    3.301222
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

If you run this command, you will see that cross-validation takes much longer than plugin. For each cross-fit, cross-validation performs its own 10-way partition of the data and runs lassos on each of those 10 partitions for the variables `react` and `no2_class`. After all this computation, the results look remarkably similar. Our coefficient estimate is still 2.3 and is still highly significant. Our 95% confidence interval is 1.4 to 3.3. This point estimate and the one obtained by plugin are close and well within each respective confidence interval.

This high degree of similarity is not always the case. Sometimes different methods produce different results.

Given that the results are so similar, you might guess that plugin and cross-validation selected similar controls. A quick glance at the header will dispel that thought. Cross-validation selected 26 controls, far more than the 10 controls selected by plugin. Remember that picking the “right” model is not what these methods are about. As long as the selected controls adequately control for everything necessary to fit the variables of interest, they are doing their job.

For these data and this model, the results simply are not very sensitive to the number of controls selected. This is true over a broad range—at the least from the 10 controls selected by plugin to the 26 controls selected by cross-validation.

Let’s take a quick look at the lassos:

```
. lassoinfo

    Estimate: active
     Command: xporegress

---------------------------------------------------------------
             |                              No. of selected variables
             |                 Selection   -------------------------
    Variable |      Model         method      min    median      max
-------------+-------------------------------------------------
   no2_class |     linear             cv        9        13       16
       react |     linear             cv        6        15       19
---------------------------------------------------------------
```

Even within cross-fits, cross-validation shows a lot more variation than plugin. The number of selected controls from the lassos on `no2_class` ranges from 9 to 16. The lassos for `react` show even more variation, ranging from 6 to 19 selected controls. Where did the 26 controls in the output of `xporegress` come from? It is a count of the union of all controls from any lasso.

Let’s peer a bit deeper into the lassos by using `lassoinfo`:

```
. lassoinfo, each

    Estimate: active
     Command: xporegress

─────────────┬─────────────────────────────────────────────────────────────
             │                                                     No. of
   Dependent │                 Selection  xfold  Selection       selected
    variable │      Model         method    no.  criterion   lambda variables
─────────────┼─────────────────────────────────────────────────────────────
   no2_class │     linear             cv      1    CV min.  .1801304       14
   no2_class │     linear             cv      2    CV min.  .2561599       10
   no2_class │     linear             cv      3    CV min.  .2181624       13
   no2_class │     linear             cv      4    CV min.  .1963854       13
   no2_class │     linear             cv      5    CV min.  .2352711       11
   no2_class │     linear             cv      6    CV min.  .2663564       12
   no2_class │     linear             cv      7    CV min.  .1293717       16
   no2_class │     linear             cv      8    CV min.  .1722497       15
   no2_class │     linear             cv      9    CV min.   .264197        9
   no2_class │     linear             cv     10    CV min.  .1184878       16
       react │     linear             cv      1    CV min.  2.130811       19
       react │     linear             cv      2    CV min.  2.443412       16
       react │     linear             cv      3    CV min.  2.062956       17
       react │     linear             cv      4    CV min.  4.220311       13
       react │     linear             cv      5    CV min.  7.434224        8
       react │     linear             cv      6    CV min.  3.356193       14
       react │     linear             cv      7    CV min.  7.954354        6
       react │     linear             cv      8    CV min.  6.422852        8
       react │     linear             cv      9    CV min.  2.982171       15
       react │     linear             cv     10    CV min.  2.738883       18
─────────────┴─────────────────────────────────────────────────────────────
```

We see that the lasso penalty parameter λ and the associated number of selected variables varies widely. This is particularly true of the lassos for `react`. It simply does not matter; the estimates for `no2_class`, our covariate of interest, are not affected.

2.4 Fitting via double selection (ds) using cross-validation

Continuing with the example above, we will fit the model using double selection and cross-validation. We recommend this for three reasons.

First, the double-selection method works quite a bit differently from the partialing out done by cross-fit. Instead of working with the lasso results one at a time and then using method of moments to estimate the parameters, double selection takes the union of selected covariates from all lassos and then just does a linear regression of `react` on `no2_class` and that union of selected covariates. The two methods are asymptotically equivalent if both sparsity bounds are met, but in finite samples, they can respond differently to any violation of the conditions required by the inferential lasso estimators. See *Solutions that focus on the true model* in [LASSO] **Lasso inference intro** for a discussion of sparsity bounds.

Second, double selection requires only two lassos for our model, making it much easier to explore the lassos.

Third, double selection is much easier to explain. We just did it above in half a sentence.

```
. dsregress react no2_class, controls($cc i.($fc)) selection(cv) rseed(12345)

Estimating lasso for react using cv
Estimating lasso for no2_class using cv

Double-selection linear model         Number of obs               =      1,036
                                      Number of controls          =         32
                                      Number of selected controls =         22
                                      Wald chi2(1)                =      24.17
                                      Prob > chi2                 =     0.0000

-----------------------------------------------------------------------------
             |               Robust
       react | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+---------------------------------------------------------------
   no2_class |   2.404191   .4890458     4.92   0.000     1.445679    3.362704
-----------------------------------------------------------------------------

Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

The coefficient estimate for `no2_class` is now to 2.4, still almost the same as fitting by `xporegress` with plugin selection. The associated confidence interval is 1.4 to 3.4. Our test against 0 was strong and is still strong. This really is a benign dataset for these linear models.

As with cross-validation, with cross-fit the number of selected controls is large—22.

What we are seeing are incredibly stable estimates.

2.5 Fitting via the other 22 methods

We will not show the results of the other 22 methods for fitting this model. Here is what you would type for each method:

```
. xporegress react no2_class, controls($cc i.($fc)) selection(adaptive) rseed(12345)

. poregress  react no2_class, controls($cc i.($fc)) rseed(12345)
. poregress  react no2_class, controls($cc i.($fc)) selection(cv) rseed(12345)
. poregress  react no2_class, controls($cc i.($fc)) selection(adaptive) rseed(12345)

. dsregress  react no2_class, controls($cc i.($fc)) rseed(12345)
. dsregress  react no2_class, controls($cc i.($fc)) selection(adaptive) rseed(12345)

. xporegress react no2_class, controls($cc i.($fc)) sqrtlasso rseed(12345)
. xporegress react no2_class, controls($cc i.($fc)) selection(cv) sqrtlasso      ///
     rseed(12345)

. poregress  react no2_class, controls($cc i.($fc)) sqrtlasso rseed(12345)
. poregress  react no2_class, controls($cc i.($fc)) selection(cv) sqrtlasso      ///
     rseed(12345)

. dsregress  react no2_class, controls($cc i.($fc)) sqrtlasso rseed(12345)
. dsregress  react no2_class, controls($cc i.($fc)) selection(cv) sqrtlasso      ///
     rseed(12345)

. xporegress react no2_class, controls($cc i.($fc)) semi rseed(12345)
. xporegress react no2_class, controls($cc i.($fc)) selection(cv) semi rseed(12345)
. xporegress react no2_class, controls($cc i.($fc)) selection(adaptive) semi     ///
     rseed(12345)

. poregress  react no2_class, controls($cc i.($fc)) semi rseed(12345)
. poregress  react no2_class, controls($cc i.($fc)) selection(cv) semi rseed(12345)
. poregress  react no2_class, controls($cc i.($fc)) selection(adaptive) semi     ///
     rseed(12345)

. xporegress react no2_class, controls($cc i.($fc)) sqrtlasso semi rseed(12345)
. xporegress react no2_class, controls($cc i.($fc)) selection(cv) sqrtlasso semi  ///
     rseed(12345)

. poregress  react no2_class, controls($cc i.($fc)) sqrtlasso semi rseed(12345)
. poregress  react no2_class, controls($cc i.($fc)) selection(cv) sqrtlasso semi  ///
     rseed(12345)
```

By now, the commands are nearly self explanatory.

Command `xporegress` fits via the cross-fit partialing-out method. Command `poregress` fits via the partialing-out method. Command `dsregress` fits via the double-selection method.

Adding option `selection(cv)` specifies that cross-validation select the covariates. Adding option `selection(adaptive)` specifies that adaptive lasso select the covariates. No `selection()` option implies that the plugin method (the default) select the covariates.

Adding option `sqrtlasso` specifies the square-root lasso rather than standard lasso.

Adding option `semi` specifies an alternate way of combining the moments for the po and xpo methods.

If you are interested, run some or all of these commands.

If you do, you will find that for these data and this model, the method we choose makes little difference. The results for these 22 methods look a lot like the results for the first 3 methods. The maximum coefficient for `no2_class` is 2.4, and the minimum coefficient is 2.3. The maximum standard error is 0.51, and the minimum is 0.48. All methods reject that the coefficient for `no2_class` is 0 well beyond the 1% level of significance.

The close similarity of the results from all 25 methods may seem surprising. Are they all selecting the same controls? The answer is no. Recall from *2.2 Fitting via cross-fit partialing out (xpo) using plugin* that the selected number of controls is 10, whereas from *2.4 Fitting via double selection (ds) using cross-validation*, the selected number of controls is 22—over twice as many.

Let's look at just two of the methods to see which controls they are selecting. We can easily do this only lasso by lasso (not command by command), so we will use two double-selection methods. Double selection creates only two lassos for our model. Comparing the cross-fit methods would require looking at 20 lassos per method. Let's use `lassocoef` to compare double selection using plugin and double selection using cross-validation.

First, we rerun those two models and store their estimates.

```
. dsregress react no2_class, controls($cc i.($fc)) rseed(12345)
  (output omitted)
. estimates store ds_plugin
. dsregress react no2_class, controls($cc i.($fc)) selection(cv) rseed(12345)
  (output omitted)
. estimates store ds_cv
```

Then, we compare the selected controls from each lasso.

```
. lassocoef (ds_plugin, for(react))
>           (ds_cv    , for(react))
>           (ds_plugin, for(no2_class))
>           (ds_cv    , for(no2_class))

------------------------------------------------------------------
                 |  ds_plugin     ds_cv     ds_plugin     ds_cv
                 |    react       react     no2_class   no2_class
-----------------+------------------------------------------------
             age |      x           x                       x
           0.sex |      x           x
                 |
           grade |
             2nd |      x           x
             4th |      x           x
             3rd |                                          x
                 |
      feducation |
      University |      x           x                       x
         Primary |                  x
        <Primary |                                          x
                 |
            age0 |                  x
        sev_home |                  x                       x
  siblings_young |                  x                       x
     0.lbweight  |                  x
                 |
      meducation |
               1 |                  x
               2 |                  x
                 |
        no2_home |                              x           x
      green_home |                              x           x
    noise_school |                              x           x
      sev_school |                              x           x
          precip |                              x           x
                 |
      breastfeed |
No breastfeeding |                                          x
       >6 months |                                          x
                 |
          msmoke |
      No smoking |                                          x
           _cons |      x           x           x           x
------------------------------------------------------------------
Legend:
  b - base level
  e - empty cell
  o - omitted
  x - estimated
```

The first two columns of x's show which controls were selected from the lassos for the dependent variable, `react`—the first column for the plugin method and the second for cross-validation. The third and fourth columns of x's show which controls were selected by the lassos for the covariate of interest, `no2_class`.

Cross-validation selected more controls than did plugin in the lassos for both the dependent variable, react, and the covariate of interest, no2_class. That is not surprising because plugin is designed to be cautious about adding noise through variable selection while cross-validation cares only about minimizing the cross-validation mean squared error.

Perhaps more interesting is that for both react and no2_class, cross-validation selected a superset of the variables selected by plugin. While not guaranteed, that result is a reflection of how the lasso works. Plugin and cross-validation select their covariates by setting an "optimal" value of λ, the lasso penalty. Plugin selects a larger λ and thereby a stronger penalty that selects fewer variables. As the penalty gets weaker, lasso can drop selected variables when adding others, but lasso is more likely to simply add variables. So, in this case, cross-validation's weaker penalty leads to a superset of the variables selected by plugin. That is a bit of an oversimplification because plugin selects variables that have been weighted by the inverse standard deviation of their scores while cross-validation does not weight the variables. This means that the lambda for plugin and the lambda for cross-validation are on different scales.

Recall, though, that the only role of the selected controls is to adequately capture the unmodeled correlations among the dependent variable, the variables of interest, and the model's error.

2.6 Fitting models with several variables of interest

All 11 inferential models in Stata allow you to have more than one variable of interest. Let's extend our base example from section 2.2 to include both no2_class and student's age as variables of interest.

The only trick is that we must remove our new variables of interest from our list of continuous controls. vl makes that easy:

```
. vl create cc6 = cc - (age)
note: $cc6 initialized with 9 variables.
```

We have now created the global macro cc6, which has the same variables as cc except that age has been removed.

We fit the model using cross-fit partialing-out with the default plugin selection by typing

```
. xporegress react no2_class age, controls($cc6 i.($fc)) rseed(12345)
```

(output omitted)

```
Cross-fit partialing-out              Number of obs                =      1,036
linear model                          Number of controls           =         31
                                      Number of selected controls  =          9
                                      Number of folds in cross-fit =         10
                                      Number of resamples          =          1
                                      Wald chi2(2)                 =      25.24
                                      Prob > chi2                  =     0.0000

------------------------------------------------------------------------------
             |               Robust
       react | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
   no2_class |   2.353826   .4892462     4.81   0.000     1.394921    3.312731
         age |  -25.01451   11.38901    -2.20   0.028    -47.33656    -2.69245
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

The coefficient for no2_class has barely changed at all from its estimate in section 2.2.

Again, we interpret the coefficients on the variables of interest just as we would if they were part of a standard linear regression. So a 1-unit change in no2_class elicits a 2.4-unit change in react. A 1-unit change in age elicits a −25-unit change in react. Because the relationship is linear, these changes can be interpreted as the expected change for any individual or as the expected change for any population or subpopulation of interest.

2.7 Fitting models with factor variables of interest

Having a factor variable of interest is really no different than having several variables of interest. Factor variables just provide a convenient way to add several indicator variables to our model.

Those who study response times for children know that they decrease (improve) as the child is exposed over time to educational stimuli. We might then be interested in how the response times vary across the child's grade level. Ignoring our original interest in the effect of nitrogen dioxide for the moment, let's pretend our only variable of interest is grade in school.

The distribution of grades in our sample looks like this:

```
. tabulate grade

   Grade in |
     school |      Freq.     Percent        Cum.
------------+-----------------------------------
        2nd |        412       37.83       37.83
        3rd |        397       36.46       74.29
        4th |        280       25.71      100.00
------------+-----------------------------------
      Total |      1,089      100.00
```

If we wish to use the levels of grade as our variables of interest, we need to remove it from our list of factor-variable controls:

```
. vl create fc7 = fc - (grade)
note: $fc7 initialized with 7 variables.
```

We are not currently interested in the effect of nitrogen dioxide, so we need to add it back to the list of continuous controls:

```
. vl create cc7 = cc + (no2_class)
note: $cc7 initialized with 11 variables.
```

We can now fit our model with the levels of grade as our variables of interest. We add the option baselevels so that we can see which level of grade has been made the base level.

```
. xporegress react i.grade, controls($cc7 i.($fc7)) baselevels rseed(12345)
  (output omitted)
Cross-fit partialing-out             Number of obs                 =      1,036
linear model                         Number of controls            =         30
                                     Number of selected controls   =          5
                                     Number of folds in cross-fit  =         10
                                     Number of resamples           =          1
                                     Wald chi2(2)                  =      16.82
                                     Prob > chi2                   =     0.0002

------------------------------------------------------------------------------
             |               Robust
       react | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
       grade |
         2nd |          0  (base)
         3rd |  -62.07497   15.26513    -4.07   0.000    -91.99408   -32.15587
         4th |  -92.52593   25.02151    -3.70   0.000    -141.5672   -43.48467
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

A common theme in these sections is that we interpret the results for the variables of interest just as we would if they were part of a linear regression. This is no different for factor variables. As we would with a simple linear regression, we interpret each of the coefficients as the increases in performance relative to the base level for second graders. We can see that mean reaction time is 62 milliseconds faster for third graders than it is for second graders. Fourth graders are, on average, 93 milliseconds faster than second graders.

That common theme extends to the tools that are available after fitting a model with any of the lasso inference commands. For example, we can use contrast to do comparisons of the grade levels that are not against a reference category, as they were in the regression. We could use a reverse adjacent (ar.) contrast to compare each grade to the prior grade:

```
. contrast ar.grade
Contrasts of marginal linear predictions
Margins: asbalanced

------------------------------------------------
              |         df        chi2     P>chi2
--------------+---------------------------------
        grade |
(3rd vs 2nd)  |          1       16.54     0.0000
(4th vs 3rd)  |          1        4.23     0.0397
        Joint |          2       16.82     0.0002
------------------------------------------------

--------------------------------------------------------------
              |   Contrast   Std. err.     [95% conf. interval]
--------------+-----------------------------------------------
        grade |
(3rd vs 2nd)  |  -62.07497   15.26513     -91.99408   -32.15587
(4th vs 3rd)  |  -30.45096   14.80702     -59.47218   -1.429727
--------------------------------------------------------------
```

The regression showed a 62-millisecond decrease in response time when comparing third graders to second graders, and that is reproduced by contrast. The difference with reverse-adjacent comparisons is that the comparison group for fourth graders is now third graders, and we estimate that difference to be a 30-millisecond decrease. It would take a bit more work to determine if the apparently slower improvement from third to fourth grade is indeed significantly different from the improvement from second to third grade. If you are interested, and without explanation, you could type

```
. contrast ar.grade, post
. lincom _b[ar2vs1.grade] - _b[ar3vs2.grade]
```

You will find that, by a slim margin, we fail to distinguish between the effect of going from second to third grade and the effect of going from third to fourth grade.

If we had a more complicated set of interest, we would find contrast indispensable. If you have factor variables of interest, we suggest you become familiar with contrast.

What we cannot do with results from the inferential lasso models is use margins to estimate population and subpopulation means. margins requires a full coefficient vector and variance matrix for those coefficients. The lasso inference commands can only tell us about a subset of that coefficient vector and associated variance matrix.

If you are epidemiologically inclined, you might wonder if the effect of grade is not just a proxy for increasing age. Now that we have multiple variables of interest and factor variables of interest, we can check that too:

```
. vl create cc7b = cc7 - (age)
note: $cc7b initialized with 10 variables.

. xporegress react age i.grade, controls($cc7b i.($fc7)) baselevels rseed(12345)
  (output omitted)
Cross-fit partialing-out               Number of obs                 =      1,036
linear model                           Number of controls            =         29
                                       Number of selected controls   =          3
                                       Number of folds in cross-fit  =         10
                                       Number of resamples           =          1
                                       Wald chi2(3)                  =     203.93
                                       Prob > chi2                   =     0.0000

------------------------------------------------------------------------------
             |               Robust
       react | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         age |  -18.50751   11.16037    -1.66   0.097    -40.38143    3.366418
             |
       grade |
        2nd  |          0  (base)
        3rd  |  -67.35294    15.4679    -4.35   0.000    -97.66947   -37.03641
        4th  |  -100.7346    25.0814    -4.02   0.000    -149.8932   -51.57594
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

The estimates for grade level have changed only a bit. Response times may improve with age, but we cannot detect that at the 5% level. Regardless, the effect of educational stimulation appears to be independent. Most importantly, we see that all of our `contrast` tools can be used with these estimators.

2.8 Fitting models with interactions of interest

Not surprisingly, tools for evaluating interactions for other estimation commands are also available to evaluate interactions among our variables of interest, whether those interactions are strictly among factor variables or are with factor variables and continuous variables. Let's arbitrarily check for an interaction between the child's sex and his or her age. Again, we need to manage our list of controls by removing `sex` and `age` from the list of factor-variable controls. And we again need to put `no2_class`, which is no longer a variable of interest, back into the continuous controls.

```
. vl create fc8 = fc - (sex grade)
note: $fc8 initialized with 6 variables.

. vl create cc8 = cc + (no2_class)
note: $cc8 initialized with 11 variables.
```

We can then fit a cross-fit model of reaction time where our variable of interest is `sex##grade`—the interaction of `sex` and `grade` while also including individual indicators for the levels of `sex` and `grade`.

```
. xporegress react sex##grade, controls($cc8 i.($fc8)) baselevels rseed(12345)
  (output omitted)
Cross-fit partialing-out          Number of obs               =      1,036
linear model                      Number of controls          =         28
                                  Number of selected controls =          6
                                  Number of folds in cross-fit =        10
                                  Number of resamples         =          1
                                  Wald chi2(5)                =      64.57
                                  Prob > chi2                 =     0.0000

------------------------------------------------------------------------------
             |               Robust
       react | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         sex |
       Male  |          0  (base)
     Female  |   45.10077   13.73912     3.28   0.001     18.17259    72.02896
             |
       grade |
        2nd  |          0  (base)
        3rd  |  -65.62381   17.72386    -3.70   0.000     -100.362   -30.88568
        4th  |  -102.2437    26.5379    -3.85   0.000     -154.257   -50.23033
             |
   sex#grade |
 Female#3rd  |   3.173242   19.09434     0.17   0.868    -34.25098    40.59747
 Female#4th  |   18.42495   19.98327     0.92   0.357    -20.74154    57.59144
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

The two coefficients of the interaction `sex#grade` and their associated statistics do not give us much hope that an interaction is statistically detectable. Let's check anyway:

```
. contrast sex#grade
Contrasts of marginal linear predictions
Margins: asbalanced

------------------------------------------------
             |         df        chi2     P>chi2
-------------+----------------------------------
   sex#grade |          2        0.96     0.6188
------------------------------------------------
```

Definitely not statistically significant, at any level.

What about the individual effects of `sex` and `grade`?

```
. contrast sex
Contrasts of marginal linear predictions
Margins: asbalanced

------------------------------------------------
             |         df        chi2     P>chi2
-------------+----------------------------------
         sex |          1       42.33     0.0000
------------------------------------------------
```

```
. contrast grade
Contrasts of marginal linear predictions
Margins: asbalanced

─────────────┬────────────────────────────────────
             │         df        chi2     P>chi2
─────────────┼────────────────────────────────────
       grade │          2       17.83     0.0001
─────────────┴────────────────────────────────────
```

Both individual effects are significant at any level you would care to consider.

Some studies have found differences in some types of reaction times between the sexes, but we might want to consider another factor—the interaction between sex and no2_class.

We can put grade back into the controls because it has no interaction with sex.

```
. vl create fc8b = fc - (sex)
note: $fc8b initialized with 7 variables.
```

We are ready to fit a model that includes sex, no2_class, and their interaction. That can be written in shorthand, by typing c.no2_class##i.sex. We fit the model:

```
. xporegress react c.no2_class##i.sex, controls($cc i.($fc8b)) rseed(12345)
  (output omitted)
Cross-fit partialing-out                     Number of obs               =      1,036
linear model                                 Number of controls          =         30
                                             Number of selected controls =          9
                                             Number of folds in cross-fit =        10
                                             Number of resamples         =          1
                                             Wald chi2(3)                =      63.42
                                             Prob > chi2                 =     0.0000

─────────────┬────────────────────────────────────────────────────────────────
             │               Robust
       react │ Coefficient  std. err.      z    P>|z|     [95% conf. interval]
─────────────┼────────────────────────────────────────────────────────────────
   no2_class │   1.708798   .5961435     2.87   0.004     .5403779    2.877217
             │
         sex │
      Female │   17.47061   24.31548     0.72   0.472    -30.18686    65.12807
             │
        sex# │
 c.no2_class │
      Female │   1.099669   .7737183     1.42   0.155    -.4167913    2.616129
─────────────┴────────────────────────────────────────────────────────────────
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

Everything we need to know is in this output.

The effect of no2_class is still positive, as it was for all of our earlier fits. The effect is now a bit smaller at a 1.7-millisecond increase in response for every microgram increase in NO_2 per cubic meter.

There is no longer a significant difference in response times for females compared with males. The point estimate is 17, but its z statistic is a scant 0.72.

The interaction between sex and no2_class is also not significant, though you might wish you had more data.

You might be curious if the effect of nitrogen dioxide across both males and females from this model is similar to our earlier models without an interaction. If we assume 50% males and 50% females, we just need to add half of the interaction term to the estimate for males.

```
. lincom no2_class + .5*c.no2_class#1.sex
 ( 1)  no2_class + .5*1.sex#c.no2_class = 0

------------------------------------------------------------------------------
       react | Coefficient  Std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         (1) |   2.258632   .4800889     4.70   0.000     1.317675    3.199589
------------------------------------------------------------------------------
```

The estimate is extremely close to the point estimate and standard errors that we obtained in *2.2 Fitting via cross-fit partialing out (xpo) using plugin*—both round to 2.3 with standard errors that round to 0.48.

While we have pretended to be performing analysis, the important thing to know is that the standard inference tools can be applied to the variables of interest.

2.9 Fitting models with a nonlinear relationship of interest

Let's continue with our reaction-time example and put a nonlinearity in no2_class into the covariates of interest. What we really mean by "nonlinear" in this context is nonlinear-but-linearizeable—polynomials, logs, ratios, and the like.

We just want to demonstrate how to think about nonlinearities with these models, so let's not dwell on where the nonlinear relationship comes from. In your work, you may have some theory or precedence for your choice of nonlinearities. For now, we know that fractional polynomials (fp) produce whole classes of reasonable curves, so we will arbitrarily pick one of those forms that allows for two inflection points—including one over the square root and the cube of the variable.

```
. generate no2fp1 = no2_class^(-2)
. generate no2fp2 = no2_class^3
```

With those as our two covariates of interest, we fit a cross-fit model. Our controls are from the model we fit in *2.2 Fitting via cross-fit partialing out (xpo) using plugin*.

```
. xporegress react no2fp1 no2fp2, controls($cc i.($fc)) rseed(12345)
 (output omitted)
Cross-fit partialing-out             Number of obs                 =     1,036
linear model                         Number of controls            =        32
                                     Number of selected controls   =        11
                                     Number of folds in cross-fit  =        10
                                     Number of resamples           =         1
                                     Wald chi2(2)                  =     24.55
                                     Prob > chi2                   =    0.0000

------------------------------------------------------------------------------
             |               Robust
       react | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
      no2fp1 |  -2915.067   2227.731    -1.31   0.191    -7281.339    1451.205
      no2fp2 |   .0005923   .0001394     4.25   0.000     .0003191    .0008655
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

We see that it is unclear if we really need two terms to model this relationship. Only one of the terms is significant. But our nonlinearity is just a construct for demonstration and we want to see how this works, so we are undeterred.

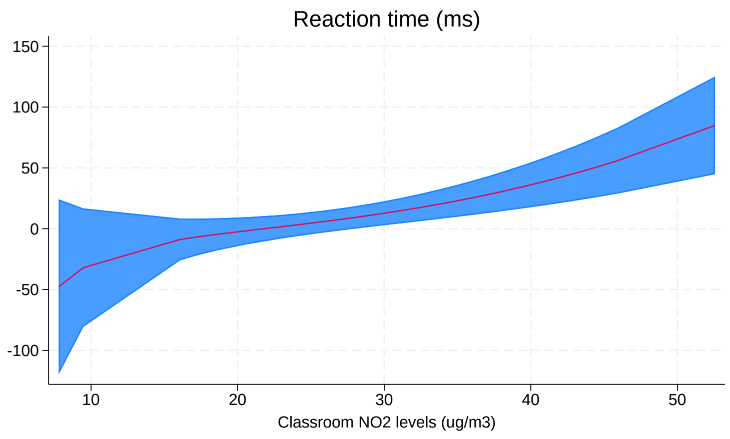
We could do a bit of algebra and decide what those terms imply about the relationship between nitrogen dioxide and reaction time. Or we could just graph the relationship. Predictions in inferential models are typically not much use, but they are perfect for our purpose.

We say predictions are not much use because the selected controls should not be used in a prediction. They are used by `xporegress` solely to obtain consistent estimates of the model of interest, but they are not themselves interpretable. So they should not be used to form predictions. We should not even use the intercept. For `xporegress` and all the other inferential models, only our covariates of interest affect the prediction. That is fine with us; that is all we want to see. We would like to get confidence intervals too, so let's use `predictnl` to get our predictions:

```
. predictnl reacthat = predict(), ci(lb ub)
note: confidence intervals calculated using Z critical values.
```

We can then graph the prediction and its confidence interval:

```
. twoway rarea lb ub no2_class, sort || line reacthat no2_class,
> sort legend(off) title("Reaction time (ms)")
```



There might be some upward curvature as nitrogen dioxide reaches its highest observed levels, but the confidence interval is too wide to be sure. The downward bend at the lowest levels of nitrogen dioxide is also suspect because the confidence interval is also wide in that region of the graph. We have scant evidence that this curve is any better than a linear relationship.

If you are unfamiliar with `twoway` syntax, we asked for two overlaid plots: a range area for the confidence interval from the variables `lb` and `ub` plotted against `no2_class`, `rarea lb ub no2_class`, and a line of predicted reaction time from the variable `reacthat` against `no2_class`.

Unfortunately, we cannot use any information-criterion tools to compare our nonlinear fit with our earlier linear fit. The inferential models cannot estimate the log likelihood or any form of residual required to form any information-criterion measures.

2.10 Controls are controls

The literature on the inferential models fit by double-selection, partialing-out, and cross-fit partialing-out estimators refers to the "variables of interest", but a more accurate term might be "submodel of interest". We say that because a maintained assumption is that the control variables are just controls and they do not interact with the variable or variables of interest. That is to say, they can shift the expected value of the outcome, but they cannot change the effect of the variables of interest.

If you think control variable `x3` actually interacts with one of your variables of interest, say, `d1`, then you will need to include that interaction in your submodel of interest. So if `x3` and `d1` are continuous, you need to add `c.x3#c.d1` to your submodel of interest; if `x3` is an indicator or multi-value factor variable, you need to add `i.x3#c.d1`; if both are factor variables, you need to add `i.x3#i.d1`. In these cases, `x3` is not a control variable—it is part of your submodel of interest.

3 Fitting logit inferential models to binary outcomes. What is different?

Even if your current interest is logit models, we suggest you also read *2 Fitting and interpreting inferential models*. That section has many more examples and goes into more detail. If you are starting here, we also suggest you read *1.4 The primary dataset* to become familiar with the dataset and how we are manipulating it. Section 1.4 is not essential reading, but if things become confusing, do read it. Here we focus primarily on what is different about logit models.

Without exception, every command and example from section 2 can be run using a logit lasso inference command. Just change `regress` to `logit` in the estimation commands, and change the dependent variable from `react` to the dependent variable we create below.

We will replicate a few of the analyses from section 2 using logit models and explain how the results are interpreted with binary outcomes. Feel free to run others. Their results are interpreted in the same way as those shown here.

Let's continue with the dataset we have been using to measure the effect of nitrogen dioxide in the classroom on the reaction time of school children.

```
. use https://www.stata-press.com/data/r19/breathe, clear
(Nitrogen dioxide and attention)
```

We need to create the global macros that will hold our lists of continuous and factor-variable control variables:

```
. do https://www.stata-press.com/data/r19/no2
  (output omitted)
```

To see how these lists were created, see *1.4 The primary dataset*.

This dataset does not have a binary (dichotomous) dependent variable, but it is easy enough to create one. The variable `omissions` contains a count of the number of times a child failed to respond to a stimuli. We can pretend that we only saw whether or not there were any omissions. Let's create a variable that is 1 when there were any omissions and is 0 otherwise:

```
. generate miss1 = omissions >= 1  if !missing(omissions)
(5 missing values generated)
```

Then take a quick look at our new variable:

```
. tabulate miss1

      miss1 |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        508       46.86       46.86
          1 |        576       53.14      100.00
------------+-----------------------------------
      Total |      1,084      100.00
```

We have 508 children who never missed a stimulus from the test and 576 who missed at least one stimulus.

3.1 Interpreting standard odds ratios

If you are new to inferential lasso models and have not at least read *2.2 Fitting via cross-fit partialing out (xpo) using plugin*, do that now. We will only explain how to interpret the odds ratios below. Section 2.2 explains more.

We can now fit a model of how classroom nitrogen dioxide levels (`no2_class`) affect whether children miss any stimuli on a reaction-time test (`miss1`). Our continuous controls are in the global macro `$cc` and our factor-variable controls are in the global macro `$fc`, as they were in our very first example in section 2.2. We use `xpologit` to fit the model:

```
. xpologit miss1 no2_class, controls($cc i.($fc)) rseed(12345)
  (output omitted)
Cross-fit partialing-out              Number of obs               =      1,036
logit model                           Number of controls          =         32
                                      Number of selected controls =          5
                                      Number of folds in cross-fit =        10
                                      Number of resamples         =          1
                                      Wald chi2(1)                =      11.18
                                      Prob > chi2                 =     0.0008

------------------------------------------------------------------------------
             |               Robust
       miss1 | Odds ratio   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
   no2_class |   1.027338   .0082854     3.34   0.001     1.011227    1.043706
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

The odds ratio for `no2_class` is 1.03. We interpret that ratio just as we would if this were a `logistic` regression. For every unit increase in the level of nitrogen dioxide, the odds of a student missing at least one stimulus increase by a factor of 1.03, with a confidence interval of 1.01 to 1.04. As always with these models, we cannot estimate a constant, so we do not know the baseline odds.

At face value, that is a small odds ratio, but the range of `no2_class` is 7.8 to 52.6:

```
. summarize no2_class

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
   no2_class |      1,089    30.16779    9.895886   7.794096   52.56397
```

The difference is over 44 micrograms per cubic meter. What odds ratio do we obtain if we increase nitrogen dioxide levels by 44?

```
. lincom _b[no2_class]*44, or
 ( 1)  44*no2_class = 0

------------------------------------------------------------------------------
       miss1 | Odds ratio   Std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         (1) |   3.276333   1.162629     3.34   0.001     1.634306    6.568144
------------------------------------------------------------------------------
```

The odds go up by 3.3 with a confidence interval from 1.6 to 6.6.

Be careful if you do this by hand. The `or` option did quite a bit of work. There are several ways to write what `lincom` did behind the scenes. One way is

$$\mathrm{OR}_{44} = 1.027338^{44}$$

This follows directly from what we said about the original odds ratio being the factor by which odds increase.

Equivalently, and what really happens behind the scenes, is

$$\mathrm{OR}_{44} = e^{\beta * 44}$$

where β is the coefficient on `no2_class`, which is the log of the odds ratio shown on the `xpologit` results. These expressions produce identical results.

We said earlier that `xpologit` cannot estimate a baseline odds. It cannot estimate any odds, only odds ratios. Even so, we might consider the degree of these effects by looking at children experiencing truly low nitrogen dioxide levels, say, below 10:

```
. table miss1 if no2_class < 10

------------------
        | Frequency
--------+---------
miss1   |
  0     |        24
  1     |        10
  Total |        34
------------------
```

That gives an odds of $10/24 = 0.42$, or roughly one child missing a stimulus for every two who respond to every stimulus. If we assume that is the starting odds for a child and then increase the nitrogen dioxide levels by 44, the odds move all the way to $3.2 \times 0.42 = 1.3$. At that level of nitrogen dioxide, almost three children miss at least one stimulus for every two who respond to every stimulus.

3.2 Interpreting models with factor variables, nonlinear relationships, and interactions

Let's run through most of the examples that we first demonstrated with linear regression. We are going to set the models up quickly. Read sections 2.6 through 2.9 for more about the models. We will use the same tools; we will just ask them to report odds ratios.

In *2.6 Fitting models with several variables of interest*, we added `age` to our covariates of interest. That means we must pull `age` from our list of continuous controls.

```
. vl create cc31 = cc - (age)
note: $cc31 initialized with 9 variables.
```

We will use different global macro names throughout this section to avoid collisions with the original examples. These globals hold the same variable lists—they just have a different name.

We fit the model:

```
. xpologit miss1 no2_class age, controls($cc31 i.($fc)) rseed(12345)
  (output omitted)
Cross-fit partialing-out               Number of obs             =      1,036
logit model                            Number of controls        =         31
                                       Number of selected controls =        7
                                       Number of folds in cross-fit =      10
                                       Number of resamples       =          1
                                       Wald chi2(2)              =      13.58
                                       Prob > chi2               =     0.0011

------------------------------------------------------------------------------
             |               Robust
       miss1 | Odds ratio   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
   no2_class |   1.048213   .0760006     0.65   0.516     .9093542    1.208275
         age |   .7922585   .0647357    -2.85   0.004     .6750174    .9298628
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

There is not much to say about the results. Interpret the odds ratios as you would any logistic model with two covariates. The odds ratio for age is 0.79 and is significant at the 5% level with a 95% confidence interval from 0.68 to 0.93. So, older children are less likely to miss a stimulus. We also note that `no2_class` is now insignificant. We are asking a lot of a binary outcome signal.

In *2.7 Fitting models with factor variables of interest*, we decided that we were interested in the effect of the grade the child was in at school and no longer interested in nitrogen dioxide.

We will set our controls to reflect this:

```
. vl create fc32 = fc - (grade)
note: $fc32 initialized with 7 variables.

. vl create cc32 = cc + (no2_class)
note: $cc32 initialized with 11 variables.
```

And we fit the xpologit model:

```
. xpologit miss1 i.grade, controls($cc32 i.($fc32)) baselevels rseed(12345)
  (output omitted)
Cross-fit partialing-out             Number of obs               =      1,036
logit model                          Number of controls          =         30
                                     Number of selected controls =          3
                                     Number of folds in cross-fit =        10
                                     Number of resamples         =          1
                                     Wald chi2(2)                =       5.51
                                     Prob > chi2                 =     0.0637

------------------------------------------------------------------------------
             |               Robust
       miss1 | Odds ratio   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
       grade |
        2nd  |          1  (base)
        3rd  |   .6371055   .1232829    -2.33   0.020     .4360134    .9309425
        4th  |   .6156729   .1974266    -1.51   0.130     .3283953    1.154259
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

The odds ratio of going from second grade, the base level, to third grade is 0.64 and is significant. The odds ratio of going from second grade to fourth grade is 0.62 and is not statistically significant at the 5% level.

These results are weaker than those for the linear model for reaction time. Even so, we forge on and use contrast to look at the grade-to-grade odds ratios. contrast knows how to exponentiate results to get odds ratios, but it is not quite as smart as lincom. We will need to tell contrast to use exponential form (eform()) and to label the results as “Odds ratio”:

```
. contrast ar.grade, eform(Odds ratio)
Contrasts of marginal linear predictions
Margins: asbalanced

------------------------------------------------
             |         df        chi2     P>chi2
-------------+----------------------------------
       grade |
(3rd vs 2nd) |          1        5.43     0.0198
(4th vs 3rd) |          1        0.02     0.8777
       Joint |          2        5.51     0.0637
------------------------------------------------

--------------------------------------------------------------
             | Odds ratio   Std. err.     [95% conf. interval]
-------------+------------------------------------------------
       grade |
(3rd vs 2nd) |   .6371055   .1232829      .4360134    .9309425
(4th vs 3rd) |   .9663594   .2149063      .6249454    1.494291
--------------------------------------------------------------
```

The first comparison is still between second and third grade. We already discussed that comparison when considering the output from xpologit. contrast reports the same odds ratio and the same p-value. The second comparison is now between third and fourth grade. The point estimate is an odds ratio of 0.97, almost 1, and it is not a significant ratio at the 5% level.

We will skip section *2.8 Fitting models with interactions of interest* because it does not offer any new tools for analyzing odds ratios. You can run that model as an inferential lasso logit model on `miss1`. Just remember to add option `eform(Odds ratio)` to any of the `contrast` commands.

In *2.9 Fitting models with a nonlinear relationship of interest*, we analyzed a nonlinear relationship between reaction time and nitrogen dioxide levels. Recall from section 2.9 that we arbitrarily chose a nonlinear representation for `no2_class` that allows for two inflection points—one over the square root of `no2_class` and one over the cube of the `no2_class`. If you have already worked through section 2.9 with your current dataset, you already have the two variables for the nonlinearity in your dataset. If not, we will need to create them.

```
. generate no2fp1 = no2_class^(-2)
. generate no2fp2 = no2_class^3
```

With these variables in place, we can fit our nonlinear relationship between `miss1` and `no2_class`.

```
. xpologit miss1 no2fp1 no2fp2, controls($cc i.($fc)) rseed(12345)
  (output omitted)
convergence not achieved
    gmm step failed to converge
r(498);
```

That did not end well. Generalized method of moments (GMM) is how `pologit` and `xpologit` combine the scores from the partialing-out process to obtain the parameter estimates for the coefficients of interest. With these data and model, GMM simply could not converge. This happens. In the other examples in this section, we have mentioned that the estimates are not as significant as they were for the linear models on reaction time from section 2. Our binary outcome variable, `miss1`, has much less information than the continuous reaction time variable.

Do we think all is lost? This is the first example of instability, so let's try a little harder. We will warn you that you can try `pologit`, but it fails with the same error.

Let's take the advice from [LASSO] **Inference requirements** and try cross-validation as our selection technique. We return to the cross-fit estimator:

```
. xpologit miss1 no2fp1 no2fp2, controls($cc i.($fc)) selection(cv) rseed(12345)
  (output omitted)
convergence not achieved
    gmm step failed to converge
r(498);
```

This is tough. We cannot even try the alternate suggestion from [LASSO] **Inference requirements** because we already said that `pologit` with plugin selection failed. We will tell you now that `pologit` with cross-validation selection also fails.

We did say earlier that double selection is more stable. Let's try `dslogit`, first with cross-validation, and store the results:

```
. dslogit miss1 no2fp1 no2fp2, controls($cc i.($fc)) selection(cv) coef
> rseed(12345)

Estimating lasso for miss1 using cv
Estimating lasso for no2fp1 using cv
Estimating lasso for no2fp2 using cv

Double-selection logit model          Number of obs               =      1,036
                                      Number of controls          =         32
                                      Number of selected controls =         23
                                      Wald chi2(2)                =      16.19
                                      Prob > chi2                 =     0.0003

------------------------------------------------------------------------------
             |               Robust
       miss1 | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
      no2fp1 |  -79.45294   41.32577    -1.92   0.055      -160.45    1.544089
      no2fp2 |   7.18e-06   2.52e-06     2.85   0.004     2.24e-06    .0000121
------------------------------------------------------------------------------

Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.

. estimates store ds_cv
```

We have estimates. There is nothing to suggest instability in these results. The coefficient on `no2fp2` is tiny, but that is the cube of `no2_class`. It needs to be a small coefficient.

What does the plugin selection method have to say?

```
. dslogit miss1 no2fp1 no2fp2, controls($cc i.($fc)) coef rseed(12345)

Estimating lasso for miss1 using plugin
Estimating lasso for no2fp1 using plugin
Estimating lasso for no2fp2 using plugin

Double-selection logit model          Number of obs               =      1,036
                                      Number of controls          =         32
                                      Number of selected controls =          5
                                      Wald chi2(2)                =      14.63
                                      Prob > chi2                 =     0.0007

------------------------------------------------------------------------------
             |               Robust
       miss1 | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
      no2fp1 |  -80.76289    39.2933    -2.06   0.040    -157.7763   -3.749442
      no2fp2 |   6.01e-06   2.35e-06     2.56   0.010     1.41e-06    .0000106
------------------------------------------------------------------------------

Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.

. estimates store ds_plugin
```

Those coefficients look similar to the ones from cross-validation selection. What is more, plugin selected only 5 controls whereas cross-validation selected 23. The double-selection results are similar over a wide range of selected controls. We stored the results from the estimators, so let's peek at the controls from the two methods by using [LASSO] **lassoinfo**:

```
. lassoinfo ds_cv ds_plugin

    Estimate: ds_cv
     Command: dslogit

                                                                 No. of
                          Selection  Selection                 selected
    Variable       Model     method  criterion      lambda    variables

       miss1       logit         cv    CV min.    .0229644            6
      no2fp1      linear         cv    CV min.    .0000249           17
      no2fp2      linear         cv    CV min.    636.8366           13

    Estimate: ds_plugin
     Command: dslogit

                                                   No. of
                          Selection              selected
    Variable       Model     method     lambda  variables

       miss1       logit     plugin     .07161          0
      no2fp1      linear     plugin   .1199154          4
      no2fp2      linear     plugin   .1199154          4
```

Cross-validation is selecting many more controls for each variable's lasso: for `miss1`, 6 versus 0; for `no2fp1`, 17 versus 4; and for `no2fp2`, 13 versus 4.

Let’s look more closely with lassocoef:

```
. lassocoef (ds_plugin, for(miss1))
>           (ds_cv    , for(miss1))
>           (ds_plugin, for(no2fp1))
>           (ds_cv    , for(no2fp1))
>           (ds_plugin, for(no2fp2))
>           (ds_cv    , for(no2fp2))

-------------------------------------------------------------------------------
             | ds_plugin   ds_cv    ds_plugin    ds_cv    ds_plugin    ds_cv
             |   miss1     miss1      no2fp1     no2fp1     no2fp2     no2fp2
-------------+-----------------------------------------------------------------
         age |               x                                           x
             |
       grade |
        2nd  |               x
        4th  |                                     x
        3rd  |                                                           x
             |
0.overweight |               x                     x
             |
  feducation |
 University  |               x                                           x
   <Primary  |                                     x                     x
    Primary  |                                     x
             |
      no2fp1 |               x
      no2fp2 |               x
    no2_home |                          x          x                     x
  green_home |                          x          x          x          x
noise_school |                          x          x          x          x
      precip |                          x          x          x          x
        age0 |                                     x
    sev_home |                                     x                     x
  sev_school |                                     x          x          x
siblings_old |                                     x
       0.sex |                                     x
             |
  breastfeed |
  <6 months  |                                     x
  >6 months  |                                     x                     x
             |
  meducation |
          1  |                                     x
          2  |                                     x
             |
      msmoke |
 No smoking  |                                                           x
    Smoking  |                                                           x
             |
       _cons |     x         x          x          x          x          x
-------------------------------------------------------------------------------
Legend:
  b - base level
  e - empty cell
  o - omitted
  x - estimated
```

A careful perusal of the x’s shows that cross-validation selected each control that plugin selected for all lassos. It also selected many more controls. We have seen this behavior before. At least we are not worried that the selection method produces different results.

We graphed the nonlinear effect of nitrogen dioxide on reaction time by using a linear model in section 2.9. The path of coefficients from a logit model do not have any interpretation. Wait! The results that we saw at the beginning of this section were interpretable. All we had to do was exponentiate the total difference from some baseline and we obtained an odds ratio. We can do that here too.

The `predict` command will give us the linear predictions for just the two fractional polynomial terms. We want a confidence interval (CI), so let's use `predictnl`:

```
. predictnl xbhat = predict(), ci(xblb xbub)
note: confidence intervals calculated using Z critical values.
```

We have no intercept, so we need to pick a level of `xbhat` whose exponential will be our baseline odds. Do we think the minimum value of nitrogen dioxide is reasonable? Or do we think that is an outlier?

```
. summarize no2_class, detail
                    Classroom NO2 levels (ug/m3)
-------------------------------------------------------------
      Percentiles      Smallest
 1%     9.474087       7.794096
 5%     16.86244       7.794096
10%      18.5384       7.794096       Obs               1,089
25%     22.81843       7.794096       Sum of wgt.       1,089

50%     29.91033                      Mean           30.16779
                        Largest       Std. dev.      9.895886
75%     36.59826       52.56397
90%     42.04398       52.56397       Variance       97.92857
95%     45.97548       52.56397       Skewness       .2082405
99%     52.52346       52.56397       Kurtosis        2.63782
```

We have five identical values of 7.8 for at least the smallest five, and they are not far from the first percentile. If we can find the linear prediction for the minimum value of `no2_class`, that would be a serviceable baseline.

```
. summarize xbhat if no2_class <= r(min)
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
       xbhat |          8   -1.326629           0  -1.326629  -1.326629
```

The `r(min)` in that expression was just a saved result from the previous `summarize` command. It contained the minimum of `no2_class`. Our linear prediction that corresponds to the minimum of `no2_class` is −1.3. That is our linear baseline. We could subtract −1.3 from our linear prediction and its bounds, but the value is stored in higher precision in `r(mean)`. Let's subtract our baseline and exponentiate the results to obtain odds ratios:

```
. generate orhat = exp(xbhat - r(mean))
. generate orlb  = exp(xblb  - r(mean))
. generate orub  = exp(xbub  - r(mean))
```

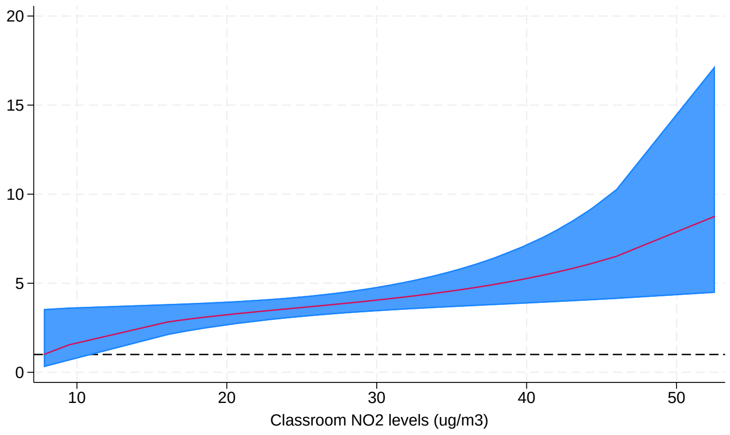
Let's label the variable holding our point estimate of the odds ratios.

```
. label variable orhat "Odds ratio vs. lowest levels of NO2"
```

It is always good to label your variables. And we would like a little labeling on our graph. If you have lost track of what we are computing, that label should be a hint.

That was a bit of work. And, admittedly, it was only loosely tied to the algebra at the top of this section. Was it worth it? What do we have?

```
. twoway rarea orlb orub no2_class, sort || line orhat no2_class,
> yline(1) legend(off) sort
```



Well, it is pretty, in a statistical way. The lowest value of the red line is exactly 1.0. It is the baseline odds that we assigned to the lowest levels of no2_class. We did that when we subtracted the prediction for the lowest levels of no2_class from all of our predictions. That made the lowest prediction exactly 0 and its exponential 1.0—meaning no effect. That was done by construction.

Let's look at the other end of the graph, the rightmost portion where no2_class levels are just above 50. The red line now looks to be between 8 and 9—we will just say 8. The odds of a child missing a stimuli when nitrogen dioxide levels are above 50 are 8 times higher than the odds when nitrogen dioxide levels are at the minimum in the dataset. For nitrogen dioxide levels of 30, the red odds-ratio line looks to be about 4, meaning that children facing levels of 30 have 4 times higher odds of missing a stimuli than do children facing the lowest levels of nitrogen dioxide. And so on. The line traces out the odds ratio for each level of nitrogen dioxide against the odds for the lowest level of nitrogen dioxide.

The blue area is the 95% confidence boundary for the odds ratio. The boundary is pretty narrow for the majority of the curve, but it expands as nitrogen dioxide levels exceed 35 or 40. At the highest levels, the band ranges from about 4 all the way to about 17.

We drew a black reference line at 1.0 because an odds ratio of 1.0 means no effect. At the lowest levels of nitrogen dioxide, the lower bound of the CI is below 1.0. So at those levels, we cannot tell whether nitrogen dioxide has an effect.

The point estimates and their CIs are in the variables orhat, orlb, and orub. You can summarize them or look at them for specific levels of no2_class.

Making the lowest level of no2_class the reference odds was arbitrary. Rather than subtract the mean of the linear prediction for that level of no2_class, we could have used the value at the mean of no2_class, or the median, or any value we choose. We need not have considered no2_class at all in setting the baseline. Any of these changes would just shift the curves up or down. Their relative positions do not change. If you have a specific comparison in mind, change the baseline.

All that said, the CIs are wide and we might be curious whether a straight line fits just as well. As we mentioned in section 2.9, the standard AIC and BIC methods for choosing among specifications are not possible after inferential lasso estimation. We are pretty much stuck with eyeing it. If you want to do that, do not try with this graph. The exponential has put its own curve onto the odds ratios. Look instead at a graph of the original predictions:

```
    twoway rarea xblb xbub no2_class, sort || line xbhat no2_class, sort
```

We leave you to draw that yourself.

4 Fitting inferential models to count outcomes. What is different?

Even if your current interest is Poisson models, we suggest you also read *2 Fitting and interpreting inferential models*. That section has many more examples and goes into more detail. If you are starting here, we also suggest you read *1.4 The primary dataset* to become familiar with the dataset and how we are manipulating it. Section 1.4 is not essential reading, but it does explain more about how we manage the variable lists in this entry. Here we focus primarily on what is different about Poisson models.

Every command and example from section 2 can be run using a Poisson lasso inference command. Just change `regress` to `poisson` in the estimation commands, and change the dependent variable from `react` to `omissions`.

We will replicate a few of the analyses from section 2 using Poisson models and explain how the results are interpreted with count outcomes. Feel free to run others. Their results are interpreted in the same way as those shown here.

Let's continue with the dataset we have been using to measure the effect of nitrogen dioxide in the classroom on the reaction time of school children.

```
    . use https://www.stata-press.com/data/r19/breathe, clear
    (Nitrogen dioxide and attention)
```

We need to create the global macros that will hold our lists of continuous and factor-variable control variables:

```
    . do https://www.stata-press.com/data/r19/no2
      (output omitted)
```

To see how these lists were created, see *1.4 The primary dataset*.

4.1 Interpreting standard incidence-rate ratios

If you are new to inferential lasso models and have not read *2.2 Fitting via cross-fit partialing out (xpo) using plugin*, do that now. We will only explain how to interpret the incident-rate ratios below. Section 2.2 explains more.

Our count outcome is `omissions`, the number of times a student failed to respond to a stimulus while taking a test to measure reaction times. We are interested in how classroom nitrogen dioxide levels (`no2_class`) affect the number of omissions.

Our continuous controls are in the global macro `$cc`, and our factor-variable controls are in the global macro `$fc`, as they were in our very first example in section 2.2. We use `xpopoisson` to fit the model,

```
. xpopoisson omissions no2_class, controls($cc i.($fc)) rseed(12345)
  (output omitted)
Cross-fit partialing-out              Number of obs               =      1,036
Poisson model                         Number of controls          =         32
                                      Number of selected controls =         16
                                      Number of folds in cross-fit =        10
                                      Number of resamples         =          1
                                      Wald chi2(1)                =       5.42
                                      Prob > chi2                 =     0.0199

------------------------------------------------------------------------------
             |               Robust
   omissions |        IRR   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
   no2_class |   1.022025   .0095654     2.33   0.020     1.003448    1.040946
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

We see that `xpopoisson` reports an IRR (incidence-rate ratio) by default, rather than a coefficient. That is more useful for interpretation. The term "rate", however, is less intuitive for the count of omissions. Often, counts are taken over a time and thus are considered rates. Our count is for a fixed-length test, so it is better to think of this as a ratio of means. Our point estimate of 1.02 means that we expect the number of omissions to go up by a factor of 1.02 for every unit increase in the level of nitrogen dioxide in the classroom. Our 95% confidence interval is 1.003 to 1.041, and the ratio is significantly different from 1 at the 5% level.

That rate might seem small, but the level of `no2_class` ranges from 7.8 to 52.6:

```
. summarize no2_class

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
   no2_class |      1,089    30.16779    9.895886   7.794096   52.56397
```

The difference is over 44 micrograms per cubic meter. A reasonable question would be how much a student is affected in going from a classroom with, say, 8 micrograms to a classroom with 52 micrograms. `lincom` can answer that question if we tell it that we want IRRs reported:

```
. lincom _b[no2_class]*44, irr

 ( 1)  44*no2_class = 0

------------------------------------------------------------------------------
   omissions |        IRR   Std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         (1) |   2.608014   1.073998     2.33   0.020     1.163535    5.845752
------------------------------------------------------------------------------
```

The ratio is 2.6 and is significant, having exactly the same z-statistic as the original estimate. That is by construction because for the purpose of the test, we merely multiplied the underlying coefficient by a constant. A child is expected to make 2.6 times as many errors when exposed to 52 micrograms of nitrogen dioxide as compared with the number of errors when exposed to only 8 micrograms.

That result does not rely on the starting number of 8. It depends only on the difference. We could ask about the effect of adding 10 micrograms of nitrogen dioxide to whatever is the ambient level:

```
. lincom _b[no2_class]*10, irr
 ( 1)  10*no2_class = 0

------------------------------------------------------------------------------
   omissions |        IRR   Std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         (1) |   1.243414   .1163742     2.33   0.020     1.035023    1.493764
------------------------------------------------------------------------------
```

So adding 10 micrograms increases the expected number of omissions by 1.24. If the number of omissions was 4 before the increase, we expect just under 5 after. If it was 10, we expect 12.4 after.

Be careful if you want to take two steps of the 10-microgram increase. These are ratios, so a 20-microgram increase leads to a $1.24^2 = 1.54$ ratio.

We cannot estimate counts after any of the Poisson inferential lasso estimators. The theory for these estimators does not provide for estimating an intercept.

4.2 Interpreting models with factor variables

As we did with logit models for binary outcomes, let's run through a few of the examples that we first demonstrated with linear regression. We are going to set the models up quickly. Read sections 2.6 and 2.7 for more about the models. We will use the same tools; we will just ask them to provide IRRs.

Continuing with the same dataset, in *2.6 Fitting models with several variables of interest* we added age to our covariates of interest. That means we must pull age from our list of continuous controls:

```
. vl create cc41 = cc - (age)
note: $cc41 initialized with 9 variables.
```

As we did with logit models, we will use different global macro names throughout this section to avoid collisions with the original examples. Again, these globals hold the same variable lists—they just have a different name.

We fit the model.

```
. xpopoisson omissions no2_class age, controls($cc41 i.($fc)) rseed(12345)
  (output omitted)
Cross-fit partialing-out            Number of obs                 =      1,036
Poisson model                       Number of controls            =         31
                                    Number of selected controls   =         15
                                    Number of folds in cross-fit  =         10
                                    Number of resamples           =          1
                                    Wald chi2(2)                  =      29.20
                                    Prob > chi2                   =     0.0000

------------------------------------------------------------------------------
             |               Robust
   omissions |        IRR   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
   no2_class |   1.023175    .005028     4.66   0.000     1.013368    1.033078
         age |   .8075872   .0406566    -4.24   0.000     .7317068    .8913366
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

We now have an IRR for age as well as for `no2_class`. They are both interpreted as we did `no2_class` above, which is to say, as you would any IRR.

In *2.7 Fitting models with factor variables of interest*, we decided that we were interested in the effect of the child's grade in school and were no longer interested in nitrogen dioxide. Really, we just want to demonstrate a factor-variable covariate of interest.

We will set our controls to reflect this:

```
. vl create fc32 = fc - (grade)
note: $fc32 initialized with 7 variables.

. vl create cc32 = cc + (no2_class)
note: $cc32 initialized with 11 variables.
```

And we fit the `xpopoisson` model:

```
. xpopoisson omissions i.grade, controls($cc32 i.($fc32)) baselevels
> rseed(12345)
  (output omitted)
Cross-fit partialing-out                    Number of obs                =     1,036
Poisson model                               Number of controls           =        30
                                            Number of selected controls  =        11
                                            Number of folds in cross-fit =        10
                                            Number of resamples          =         1
                                            Wald chi2(2)                 =      4.74
                                            Prob > chi2                  =    0.0933

------------------------------------------------------------------------------
             |               Robust
   omissions |        IRR   std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
       grade |
         2nd |          1  (base)
         3rd |   .6008938   .1451159   -2.11   0.035     .3743109    .9646349
         4th |    .443883   .1832475   -1.97   0.049      .197637    .9969392
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

The expected number of omissions of third graders is 60% of that of second graders with a 95% CI of 0.37 to 0.96. Fourth graders have even fewer omissions. The point estimate is 44% of the number for second graders.

contrast works with IRRs just as it did with ORs in section 3.2. Again, we just need to add the option eform(IRR).

```
. contrast ar.grade, eform(IRR)
Contrasts of marginal linear predictions
Margins: asbalanced

──────────────────────────────────────────────
             │         df        chi2     P>chi2
─────────────┼────────────────────────────────
       grade │
(3rd vs 2nd) │          1        4.45     0.0349
(4th vs 3rd) │          1        1.21     0.2708
      Joint  │          2        4.74     0.0933
──────────────────────────────────────────────

──────────────────────────────────────────────────────────────
             │        IRR   Std. err.     [95% conf. interval]
─────────────┼────────────────────────────────────────────────
       grade │
(3rd vs 2nd) │   .6008938   .1451159      .3743109    .9646349
(4th vs 3rd) │   .7387046   .2031553      .4309005    1.266382
──────────────────────────────────────────────────────────────
```

We specified reverse-adjacent (ar) contrasts, so comparisons will now be grade to grade rather than against a base grade. The first comparison is still between second and third grades and, of course, gives the same results as xpopoisson itself.

The second comparison is between third and fourth grades. We fail to find a significant difference, though the point estimate is that fourth graders make only 74% of the omissions made by third graders.

As with the logit models, we will skip section *2.8 Fitting models with interactions of interest* because it does not offer any new tools for analyzing odds ratios. You can run that model as an inferential lasso probit model on omissions. If you run any contrasts, be sure to add option eform(IRR).

5 Exploring inferential model lassos

Aside from the two commands we have used in the examples in this entry, [LASSO] **lassoinfo** and [LASSO] **lassocoef**, you are unlikely to need many of the postestimation commands commonly used after lasso. Regardless, most of them are available. You can create knot tables of coefficient selection, plot cross-validation functions, plot coefficient paths, display lasso coefficients, and even change the penalty parameter λ that is used to select controls.

See [LASSO] **lasso inference postestimation** for an overview and a list of postestimation commands that are available after the inferential lasso estimators. The entries for each command have examples that demonstrate their use after inferential lasso estimators.

6 Fitting an inferential model with endogenous covariates

We will replicate a well-known model that was used to illustrate a two-stage least squares estimator for handling an endogenous covariate; see Wooldridge (2010, ex. 5.3). Because the inferential lasso estimators provide variable selection that is robust to selection mistakes, we will introduce a flexible series expansion of the variables.

Wooldridge models the log of married women's wages (lwage) as a function of their experience (exper), the square of their experience, and their years of education (educ). Collectively, these are called exogenous covariates.

As is customary, education is treated as an endogenous variable. The reasoning is that we cannot measure innate ability, and ability is likely to influence both education level and income. Some disciplines refer to this as unobserved confounding rather than endogeneity. Either way, you cannot just run a regression of wages on education and experience and learn anything about the true effect of education on wages.

You need more information from variables that you presume are not affected by the woman's unmeasured ability—let's call them instruments. And, they also cannot belong in the model for wages. Wooldridge used their mothers' education (`motheduc`), their fathers' education (`fatheduc`), and their husbands' education (`huseduc`) as instruments for the woman's education. The instruments are also required to be exogenous, but we will just call them instruments.

The data are from Mroz (1987).

`xpoivregress` and `poivregress` use lassos to select the exogenous covariates from a list of potential exogenous covariates. They use lassos to select the instruments from a set of potential instruments. This means we do not have to worry about introducing noise or weak instruments by possibly including irrelevant exogenous covariates or instruments. Lasso will ensure that sufficient amounts of irrelevant covariates are ignored. We are free to include the kitchen sink.

Let's add some variables that Wooldridge kept out. He was required to be thoughtful of introducing irrelevant covariates. We are not. To the list of potential exogenous covariates, we add the number of children younger than 6 (`kidslt6`), the number of children aged 6 or older (`kidsge6`), the women's ages (`age`), their husbands' ages (`husage`), and an indicator for living in an urban area (`citt`). We have nothing to add to the instruments. Good instruments are hard to find.

To make sure the sink is full, let's take all the exogenous variables and, instead of entering them only linearly, enter them as linear terms, as quadratic terms, and as all possible interactions. Let's do the same for our list of three instruments. This is often called a series expansion, or a Taylor-series expansion. It allows for nonlinearity in the way our exogenous covariates affect the outcome and in the way our instruments control endogeneity. We just did second-order expansion; you can go further.

We will continue using the variable-management tool `vl` to manage our lists of variables. First, we use the Mroz dataset and then create our base list of exogenous covariates and our base list of instruments.

```
. use https://www.stata-press.com/data/r19/mroz, clear

. vl create exogbase = (exper age husage kidslt6 kidsge6 city)
note: $exogbase initialized with 6 variables.

. vl create instbase = (motheduc fatheduc huseduc)
note: $instbase initialized with 3 variables.
```

The list of exogenous covariates is now in the global macro `$exogbase`, and the list of instruments is now in `$instbase`.

With these base lists in hand, we can perform our expansions to create flexible nonlinear forms:

```
. vl substitute exog = c.exogbase c.exogbase#c.exogbase
. vl substitute inst = c.instbase c.instbase#c.instbase
```

The # is the factor-variable operator for interaction. It can interact categorical variables, continuous variables, or both. We could have used it directly on our estimation command line, but those lines are already long enough. We also would have to handle macro expansion by typing `$exogbase` and such. `vl` already knows about `exogbase` and `instbase` and knows to handle them as lists. The `c.` prefix tells the # operator to treat the lists as continuous variables. # assumes categorical variables unless told otherwise.

Putting it all together, `c.exogbase` means to enter all the potential exogenous covariates as themselves (linearly). `c.exogbase#c.exogbase` means to enter all possible interactions of the variables. Because an interaction of a variable with itself is a quadratic, the quadratic (squared) terms get created as part of the expansion.

Let's look at the smaller of these two lists so that we can see what we have created:

```
. macro list inst
inst:           motheduc fatheduc huseduc c.motheduc#c.motheduc
                c.motheduc#c.fatheduc c.motheduc#c.huseduc
                c.fatheduc#c.fatheduc c.fatheduc#c.huseduc c.huseduc#c.huseduc
```

That is not too bad. We count nine terms—three linear terms and six interactions (including quadratic terms).

Macro `exog` has 27 terms.

Imagine what a third-order expansion would look like. You can run into the thousands of terms quickly.

Now we can use xpoivregress to estimate the coefficient on the endogenous variable educ. We start with the plugin method to select the covariates. We do not have to specify plugin because it is the default. Specifying the rest of the model is easy because of the macro we created:

```
. xpoivregress lwage (educ = $inst), controls($exog) rseed(12345)
Cross-fit fold 1 of 10 ...
Estimating lasso for lwage using plugin
Estimating lasso for educ using plugin
Cross-fit fold 2 of 10 ...
Estimating lasso for lwage using plugin
Estimating lasso for educ using plugin
  (output omitted)
Cross-fit partialing-out           Number of obs                =        428
IV linear model                    Number of controls           =         27
                                   Number of instruments        =          9
                                   Number of selected controls  =          4
                                   Number of selected instruments =        3
                                   Number of folds in cross-fit =         10
                                   Number of resamples          =          1
                                   Wald chi2(1)                 =      10.84
                                   Prob > chi2                  =     0.0010

------------------------------------------------------------------------------
             |               Robust
       lwage | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
        educ |   .0727853   .0221045     3.29   0.001     .0294612    .1161094
------------------------------------------------------------------------------
Endogenous: educ
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

In the header, we see that 4 of 27 controls were selected, and 3 of 9 possible instruments were selected. This is a sparse representation of the model.

We estimate that every year of education increases the log of wages by 0.073. Because wages are logged, we interpret that as a rate of change, so each year of education increases wages by 7.3%. That is close to Wooldridge's estimate of 8%, and his estimate is well within our 95% CI of 2.8% to 11.6%.

Let's see how the results compare if we select using cross-validation:

```
. xpoivregress lwage (educ = $inst), controls($exog) selection(cv) rseed(12345)

Cross-fit fold 1 of 10 ...
Estimating lasso for lwage using cv
Estimating lasso for educ using cv

Cross-fit fold 2 of 10 ...
Estimating lasso for lwage using cv
  (output omitted)

Cross-fit partialing-out           Number of obs                =        428
IV linear model                    Number of controls           =         27
                                   Number of instruments        =          9
                                   Number of selected controls  =         20
                                   Number of selected instruments =        7
                                   Number of folds in cross-fit =         10
                                   Number of resamples          =          1
                                   Wald chi2(1)                 =       7.68
                                   Prob > chi2                  =     0.0056

------------------------------------------------------------------------------
             |               Robust
       lwage | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
        educ |   .0645424   .0232832     2.77   0.006     .0189082    .1101765
------------------------------------------------------------------------------
Endogenous: educ
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

Cross-validation selected 20 controls compared with the 4 selected by plugin. It selected 7 instruments compared with the 3 selected by plugin. Our point estimate of the change in wages for each additional year of education is 6.5% with a CI of 1.9% to 11.0%. The coefficient estimate from both cross-validation and plugin are significant at the 5% level. Despite having slightly different coefficient estimates, plugin and cross-validation lead to the same inferences.

References

Mroz, T. A. 1987. The sensitivity of an empirical model of married women's hours of work to economic and statistical assumptions. *Econometrica* 55: 765–799. https://doi.org/10.2307/1911029.

Sunyer, J., E. Suades-González, R. García-Esteban, I. Rivas, J. Pujol, M. Alvarez-Pedrerol, J. Forns, X. Querol, and X. Basagaña. 2017. Traffic-related air pollution and attention in primary school children: Short-term association. *Epidemiology* 28: 181–189. https://doi.org/10.1097/EDE.0000000000000603.

Wooldridge, J. M. 2010. *Econometric Analysis of Cross Section and Panel Data*. 2nd ed. Cambridge, MA: MIT Press.

Also see

[LASSO] **Lasso intro** — Introduction to lasso

[LASSO] **Lasso inference intro** — Introduction to inferential lasso models

Inference requirements — Requirements for inference

Description

The ds, po, and xpo commands, like other estimation procedures, require certain conditions be met so that their inferential results are valid. In addition, the plugin and CV selection methods have distinct properties and may perform differently under some conditions.

Remarks and examples

We assume you have read [LASSO] **Lasso inference intro**.

We fit a model with, for example, dsregress with the default plugin selection method, and then we refit the model using CV. We get slightly different results. Which is correct?

Plugin and CV are more than just different numerical techniques for model estimation. They make different assumptions, have different requirements, and have different properties. Asking which is correct has only one answer. Each is correct when their assumptions and requirements are met.

In terms of practical advice, we have two alternative recommendations.

The first one involves lots of computer time.

1. Fit the model with xpo and the default plugin.
2. Fit the model with xpo and CV.
3. Compare results. If they are similar, use the results from step 1.

This alternative will save computer time.

1. Fit the model with ds with the default plugin.
2. Fit it again with ds but with CV.
3. Fit it again with po with the default plugin.
4. Fit it again with po but with CV.
5. Compare results. If they are similar, you are likely on solid ground. If so, perform step 6.
6. Fit the model again with xpo with the default plugin and use those results.

You can combine these two recommendations. Start with the alternative, and if it fails at step 5, follow the first set of recommendations.

Also see

[LASSO] **Lasso inference intro** — Introduction to inferential lasso models

[LASSO] **lasso** — Lasso for prediction and model selection

lasso — Lasso for prediction and model selection

Description	Quick start	Menu	Syntax	Options
Remarks and examples	Stored results	Methods and formulas	References	Also see

Description

`lasso` selects covariates and fits linear, logistic, probit, Poisson, and Cox proportional hazards models. Results from `lasso` can be used for prediction and model selection.

`lasso` saves but does not display estimated coefficients. The postestimation commands listed in [LASSO] **lasso postestimation** can be used to generate predictions, report coefficients, and display measures of fit.

For an introduction to lasso, see [LASSO] **Lasso intro**.

For a description of the lasso-fitting procedure, see [LASSO] **lasso fitting**.

Quick start

Fit a linear model for `y1`, and select covariates from `x1` to `x100` using cross-validation (CV)

```
lasso linear y1 x1-x100
```

Same as above, but force `x1` and `x2` to be in the model while `lasso` selects from `x3` to `x100`

```
lasso linear y1 (x1 x2) x3-x100
```

Same as above, but fit an adaptive lasso with three steps

```
lasso linear y1 (x1 x2) x3-x100, selection(adaptive, steps(3))
```

Fit a logistic model for binary outcome `y2`, and set a random-number seed for reproducibility

```
lasso logit y2 x1-x100, rseed(1234)
```

Fit a Poisson model for count outcome `y3` with exposure `time`

```
lasso poisson y3 x1-x100, exposure(time) rseed(1234)
```

Calculate the CV function beyond the CV minimum to get the full coefficient paths, knots, etc.

```
lasso linear y2 x1-x100, selection(cv, alllambdas)
```

Turn off the early stopping rule, and iterate over λ's until a minimum is found or until the end of the λ grid is reached

```
lasso linear y1 x1-x100, stop(0)
```

Same as above, but extend the λ grid to smaller values

```
lasso linear y1 x1-x100, stop(0) grid(100, ratio(1e-5))
```

Fit a Cox proportional hazards model for `t` with failure indicator `fail`, and select covariates from `x1` to `x100` using CV

```
stset t, failure(fail)
lasso cox x1-x100
```

Same as above, but select covariates by minimizing the Bayesian information criterion (BIC) function

```
lasso cox x1-x100, selection(bic)
```

Menu

Statistics > Lasso > Lasso

Syntax

For linear, logit, probit, and Poisson models

`lasso` *model* *depvar* [(*alwaysvars*)] *othervars* [*if*] [*in*] [*weight*] [, *options*]

For Cox models

`lasso cox` [(*alwaysvars*)] *othervars* [*if*] [*in*] [, *options*]

model is one of `linear`, `logit`, `probit`, or `poisson`.

alwaysvars are variables that are always included in the model.

othervars are variables that `lasso` will choose to include in or exclude from the model.

options	Description
Model	
* `noconstant`	suppress constant term
`selection(`*sel_method*`)`	selection method to select a value of the lasso penalty parameter λ^* from the set of possible λ's
`offset(`*varname*$_o$`)`	include *varname*$_o$ in model with coefficient constrained to 1
`exposure(`*varname*$_e$`)`	include ln(*varname*$_e$) in model with coefficient constrained to 1 (`poisson` model only)
* `cluster(`*clustvar*`)`	specify cluster variable *clustvar*
Optimization	
[`no`]`log`	display or suppress an iteration log
`rseed(`#`)`	set random-number seed
`grid(`#$_g$ [, `ratio(`#`)` `min(`#`)`]`)`	specify the set of possible λ's using a logarithmic grid with #$_g$ grid points
`stop(`#`)`	tolerance for stopping the iteration over the λ grid early
`cvtolerance(`#`)`	tolerance for identification of the CV function minimum
`bictolerance(`#`)`	tolerance for identification of the BIC function minimum
`tolerance(`#`)`	convergence tolerance for coefficients based on their values
`dtolerance(`#`)`	convergence tolerance for coefficients based on deviance
`penaltywt(`*matname*`)`	programmer's option for specifying a vector of weights for the coefficients in the penalty term

sel_method	Description
`cv` [, *cv_opts*]	select λ^* using CV; the default
`adaptive` [, *adapt_opts cv_opts*]	select λ^* using an adaptive lasso
* `plugin` [, *plugin_opts*]	select λ^* using a plugin iterative formula
`bic` [, *bic_opts*]	select λ^* using BIC function
`none`	do not select λ^*

cv_opts	Description
`folds(#)`	use # folds for CV
`alllambdas`	fit models for all λ's in the grid or until the `stop(#)` tolerance is reached; by default, the CV function is calculated sequentially by λ, and estimation stops when a minimum is identified
`serule`	use the one-standard-error rule to select λ^*
`stopok`	when the CV function does not have an identified minimum and the `stop(#)` stopping criterion for λ was reached at λ_{stop}, set the selected λ^* to be λ_{stop}; the default
`strict`	do not select λ^* when the CV function does not have an identified minimum; this is a stricter alternative to the default `stopok`
`gridminok`	when the CV function does not have an identified minimum and the `stop(#)` stopping criterion for λ was not reached, set the selected λ^* to be the minimum of the λ grid, λ_{gmin}; this is a looser alternative to the default `stopok` and is rarely used

adapt_opts	Description
`steps(#)`	use # adaptive steps (counting the initial lasso as step 1)
`unpenalized`	use the unpenalized estimator to construct initial weights
`ridge`	use the ridge estimator to construct initial weights
`power(#)`	raise weights to the #th power

plugin_opts	Description
`heteroskedastic`	assume model errors are heteroskedastic; the default
`homoskedastic`	assume model errors are homoskedastic

bic_opts	Description
alllambdas	fit models for all λ's in the grid or until the stop(#) tolerance is reached; by default, the BIC function is calculated sequentially by λ, and estimation stops when a minimum is identified
stopok	when the BIC function does not have an identified minimum and the stop(#) stopping criterion for λ was reached at λ_{stop}, set the selected λ^* to be λ_{stop}; the default
strict	do not select λ^* when the BIC function does not have an identified minimum; this is a stricter alternative to the default stopok
gridminok	when the BIC function does not have an identified minimum and the stop(#) stopping criterion for λ was not reached, set the selected λ^* to be the minimum of the λ grid, λ_{gmin}; this is a looser alternative to the default stopok and is rarely used
postselection	use postselection coefficients to compute BIC

*noconstant, cluster(), and selection(plugin) are not allowed with lasso cox.

You must stset your data before using lasso cox; see [ST] **stset**.

alwaysvars and *othervars* may contain factor variables; see **[U] 11.4.3 Factor variables**.

collect is allowed; see **[U] 11.1.10 Prefix commands**.

Default weights are not allowed. iweights are allowed with all *sel_method* options. fweights are allowed when selection(plugin), selection(bic), or selection(none) is specified. See **[U] 11.1.6 weight**. For lasso cox, weights must be specified when you stset your data.

penaltywt(*matname*) does not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Lasso estimation consists of three steps that the software performs automatically. Understanding the steps is important for understanding how to specify options. A grid for λ is used for selection methods cv, adaptive, bic, and none. selection(adaptive) resets the grid in the second and subsequent lassos. selection(plugin) bypasses steps 1 and 2. It does not require a λ grid.

Step 1: Set λ grid

A grid for λ is set. Either the default grid can be used or grid options can be specified to modify the default. The maximum λ in the grid is λ_{gmax}. It is automatically set to the smallest λ yielding a model with all coefficients zero. The minimum λ in the grid is λ_{gmin}. Typically, estimation ends before λ_{gmin} is reached when a minimum of the CV or BIC function is found. If λ_{gmin} is reached without finding a minimum, you may want to make λ_{gmin} smaller. You can do this by setting λ_{gmin} or, alternatively, by setting the ratio $\lambda_{\text{gmin}}/\lambda_{\text{gmax}}$ to a smaller value. See the grid() option below.

Step 2: Fit the model for next λ in grid

For each λ in the grid, the set of nonzero coefficients is estimated. Estimation starts with λ_{gmax} and iterates toward λ_{gmin}. The iteration stops when a minimum of the CV or BIC function is found, the stop(#) stopping tolerance is met, or λ_{gmin} is reached. When the deviance changes by less than a relative difference of stop(#), the iteration over λ ends. To turn off this stopping rule, specify stop(0). See the optimization options below.

Step 3: Select λ^*

A λ denoted by λ^* is selected. selection(*sel_method*) specifies the method used to select λ^*. The allowed *sel_method*s are cv (the default), adaptive, plugin, bic, and none:

cv, the default, uses CV to select λ^*. After a model is fit for each λ, the CV function is computed. If a minimum of the CV function is identified, iteration over the λ grid ends. To compute the CV function for additional λ's past the minimum, specify the suboption alllambdas. When you specify this option, step 2 is first done for all λ's until the stopping tolerance is met or the end of the grid is reached. Then, the CV function is computed for all λ's and searched for a minimum. See the suboptions for selection(cv) below.

adaptive also uses CV to select λ^*, but multiple lassos are performed. In the first lasso, a λ^* is selected, and penalty weights are constructed from the coefficient estimates. Then, these weights are used in a second lasso where another λ^* is selected. By default, two lassos are performed, but more can be specified. See the suboptions for selection(adaptive) below.

plugin computes λ^* based on an iterative formula. Coefficient estimates are obtained only for this single value of λ.

bic selects λ^* by using the BIC function. It selects λ^* with the minimum BIC function value.

none does not select a λ^*. Neither the CV function nor the BIC function is computed. Models are fit for all λ's until the stopping tolerance is met or the end of the grid is reached. lasso postestimation commands can be used to assess different λ's and select λ^*.

A longer description of the lasso-fitting procedure is given in [LASSO] **lasso fitting**.

Model

noconstant omits the constant term. Note, however, when there are factor variables among the *othervars*, lasso can potentially create the equivalent of the constant term by including all levels of a factor variable. This option is likely best used only when all the *othervars* are continuous variables and there is a conceptual reason why there should be no constant term. This option is not allowed with lasso cox.

selection(cv), selection(adaptive), selection(plugin), selection(bic), and selection(none) specify the selection method used to select λ^*. These options also allow suboptions for controlling the specified selection method. selection(plugin) is not allowed with lasso cox.

selection(cv [, *cv_opts*]) is the default. It selects λ^* to be the λ that gives the minimum of the CV function. It is widely used when the goal is prediction. lasso postestimation commands can be used after selection(cv) to assess alternative λ^* values.

cv_opts are folds(#), alllambdas, serule, stopok, strict, and gridminok.

folds(#) specifies that CV with # folds be done. The default is folds(10).

alllambdas specifies that models be fit for all λ's in the grid or until the stop(#) tolerance is reached. By default, models are calculated sequentially from largest to smallest λ, and the CV function is calculated after each model is fit. If a minimum of the CV function is found, the computation ends at that point without evaluating additional smaller λ's.

alllambdas computes models for these additional smaller λ's. Because computation time is greater for smaller λ, specifying alllambdas may increase computation time manyfold. Specifying alllambdas is typically done only when a full plot of the CV function is wanted for assurance that a true minimum has been found. Regardless of whether alllambdas is specified, the selected λ^* will be the same.

`serule` selects λ^* based on the "one-standard-error rule" recommended by Hastie, Tibshirani, and Wainwright (2015, 13–14) instead of the λ that minimizes the CV function. The one-standard-error rule selects the largest λ for which the CV function is within a standard error of the minimum of the CV function.

`stopok`, `strict`, and `gridminok` specify what to do when the CV function does not have an identified minimum. A minimum is identified at λ^* when the CV function at both larger and smaller adjacent λ's is greater than it is at λ^*. When the CV function has an identified minimum, these options all do the same thing: the selected λ^* is the λ that gives the minimum. In some cases, however, the CV function declines monotonically as λ gets smaller and never rises to identify a minimum. When the CV function does not have an identified minimum, `stopok` and `gridminok` make alternative selections for λ^*, and `strict` makes no selection. You may specify only one of `stopok`, `strict`, or `gridminok`; `stopok` is the default if you do not specify one. With each of these options, estimation results are always left in place, and alternative λ^* can be selected and evaluated.

`stopok` specifies that when the CV function does not have an identified minimum and the `stop(#)` stopping tolerance for λ was reached, the selected λ^* is λ_{stop}, the λ that met the stopping criterion. λ_{stop} is the smallest λ for which coefficients are estimated, and it is assumed that λ_{stop} has a CV function value close to the true minimum. When no minimum is identified and the `stop(#)` criterion is not met, an error is issued.

`strict` requires the CV function to have an identified minimum, and if not, an error is issued.

`gridminok` is a rarely used option that specifies that when the CV function has no identified minimum and the `stop(#)` stopping criterion was not met, λ_{gmin}, the minimum of the λ grid, is the selected λ^*.

The `gridminok` selection criterion is looser than the default `stopok`, which is looser than `strict`. With `strict`, only an identified minimum is selected. With `stopok`, either the identified minimum or λ_{stop} is selected. With `gridminok`, either the identified minimum or λ_{stop} or λ_{gmin} is selected, in this order.

`selection(adaptive` [`,` *adapt_opts cv_opts*]`)` selects λ^* using the adaptive lasso selection method. It consists of multiple lassos with each lasso step using CV. Variables with zero coefficients are discarded after each successive lasso, and variables with nonzero coefficients are given penalty weights designed to drive small coefficient estimates to zero in the next step. Hence, the final model typically has fewer nonzero coefficients than a single lasso. The adaptive method has historically been used when the goal of lasso is model selection. As with `selection(cv)`, lasso postestimation commands can be used after `selection(adaptive)` to assess alternative λ^*.

adapt_opts are `steps(#)`, `unpenalized`, `ridge`, and `power(#)`.

`steps(#)` specifies that adaptive lasso with # lassos be done. By default, # = 2. That is, two lassos are run. After the first lasso estimation, terms with nonzero coefficients β_i are given penalty weights equal to $1/|\beta_i|$, terms with zero coefficients are omitted, and a second lasso is estimated. Terms with small coefficients are given large weights, making it more likely that small coefficients become zero in the second lasso. Setting $\# > 2$ can produce more parsimonious models. See *Methods and formulas*.

`unpenalized` specifies that the adaptive lasso use the unpenalized estimator to construct the initial weights in the first lasso. This option is useful when CV cannot find a minimum. `unpenalized` cannot be specified with `ridge`.

`ridge` specifies that the adaptive lasso use the ridge estimator to construct the initial weights in the first lasso. `ridge` cannot be specified with `unpenalized`.

`power(#)` specifies that the adaptive lasso raise the weights to the #th power. The default is `power(1)`. The specified power must be in the interval $[0.25, 2]$.

cv_options are all the suboptions that can be specified for `selection(cv)`, namely, `folds(#)`, `alllambdas`, `serule`, `stopok`, `strict`, and `gridminok`. The options `alllambdas`, `strict`, and `gridminok` apply only to the first lasso estimated. For second and subsequent lassos, `gridminok` is the default. When `ridge` is specified, `gridminok` is automatically used for the first lasso.

`selection(plugin` [, *plugin_opts*]`)` selects λ^* based on a "plugin" iterative formula dependent on the data. The plugin method was designed for lasso inference methods and is useful when using `lasso` to manually implement inference methods, such as double-selection lasso. The plugin estimator calculates a value for λ^* that dominates the noise in the estimating equations, which makes it less likely to include variables that are not in the true model. See *Methods and formulas*. This option is not allowed with `lasso cox`.

`selection(plugin)` does not estimate coefficients for any other values of λ, so it does not require a λ grid, and none of the grid options apply. It is much faster than the other selection methods because estimation is done only for a single value of λ. It is an iterative procedure, however, and if the plugin is computing estimates for a small λ (which means many nonzero coefficients), the estimation can still be time consuming. Because estimation is done only for one λ, you cannot assess alternative λ^* as the other selection methods allow.

plugin_opts are `heteroskedastic` and `homoskedastic`.

`heteroskedastic` (`linear` models only) assumes model errors are heteroskedastic. It is the default. Specifying `selection(plugin)` for `linear` models is equivalent to specifying `selection(plugin, heteroskedastic)`.

`homoskedastic` (`linear` models only) assumes model errors are homoskedastic. See *Methods and formulas*.

`selection(bic` [, *bic_opts*]`)` selects λ^* by using the BIC function. It selects the λ^* with the minimum BIC function value.

bic_opts are `alllambdas`, `stopok`, `strict`, `gridminok`, and `postselection`.

`alllambdas` specifies that models be fit for all λ's in the grid or until the `stop(#)` tolerance is reached. By default, models are calculated sequentially from largest to smallest λ, and the BIC function is calculated after each model is fit. If a minimum of the BIC function is found, the computation ends at that point without evaluating additional smaller λ's.

`alllambdas` computes models for these additional smaller λ's. Because computation time is greater for smaller λ, specifying `alllambdas` may increase computation time manyfold. Specifying `alllambdas` is typically done only when a full plot of the BIC function is wanted for assurance that a true minimum has been found. Regardless of whether `alllambdas` is specified, the selected λ^* will be the same.

`stopok`, `strict`, and `gridminok` specify what to do when the BIC function does not have an identified minimum. A minimum is identified at λ^* when the BIC function at both larger and smaller adjacent λ's is greater than it is at λ^*. When the BIC function has an identified minimum, these options all do the same thing: the selected λ^* is the λ that gives the minimum. In some cases, however, the BIC function declines monotonically as λ gets smaller and never rises to identify a minimum. When the BIC function does not have an identified minimum, `stopok` and `gridminok` make alternative selections for λ^*, and `strict` makes no selection. You may specify only one of `stopok`, `strict`, or `gridminok`; `stopok` is the default if you do not specify one. With each of these options, estimation results are always left in place, and alternative λ^* can be selected and evaluated.

`stopok` specifies that when the BIC function does not have an identified minimum and the `stop(#)` stopping tolerance for λ was reached, the selected λ^* is λ_{stop}, the λ that met the stopping criterion. λ_{stop} is the smallest λ for which coefficients are estimated, and it is assumed that λ_{stop} has a BIC function value close to the true minimum. When no minimum is identified and the `stop(#)` criterion is not met, an error is issued.

`strict` requires the BIC function to have an identified minimum, and if not, an error is issued.

`gridminok` is a rarely used option that specifies that when the BIC function has no identified minimum and the `stop(#)` stopping criterion was not met, then λ_{gmin}, the minimum of the λ grid, is the selected λ^*.

The `gridminok` selection criterion is looser than the default `stopok`, which is looser than `strict`. With `strict`, only an identified minimum is selected. With `stopok`, either the identified minimum or λ_{stop} is selected. With `gridminok`, either the identified minimum or λ_{stop} or λ_{gmin} is selected, in this order.

`postselection` specifies to use the postselection coefficients to compute the BIC function. By default, the penalized coefficients are used.

`selection(none)` does not select a λ^*. Lasso is estimated for the grid of values for λ, but no attempt is made to determine which λ should be selected. The postestimation command `lassoknots` can be run to view a table of λ's that define the knots (the sequential sets of nonzero coefficients) for the estimation. The `lassoselect` command can be used to select a value for λ^*, and `lassogof` can be run to evaluate the prediction performance of λ^*.

When `selection(none)` is specified, neither the CV function nor the BIC function is computed. If you want to view the knot table with values of the CV function shown and then select λ^*, you must specify `selection(cv)`. Similarly, if you want to view the knot table with values of the BIC function shown, you must specify `selection(bic)`. There are no suboptions for `selection(none)`.

`offset(`*varname*$_o$`)` specifies that *varname*$_o$ be included in the model with its coefficient constrained to be 1.

`exposure(`*varname*$_e$`)` can be specified only for the `poisson` model. It specifies that ln(*varname*$_e$) be included in the model with its coefficient constrained to be 1.

`cluster(`*clustvar*`)` specifies the cluster variable *clustvar*. Specifying a cluster variable will affect how the log-likelihood function is computed and the sample split in cross-validation. The log-likelihood function is computed as the sum of the log likelihood at the cluster levels. If option `selection(cv)` is specified, the cross-validation sample is split by the clusters defined by *clustvar*. That is, the subsample in each fold is drawn on the cluster level. Therefore, all observations in a cluster are kept together in the same subsample. This option is not allowed with `lasso cox`.

Optimization

[`no`]`log` displays or suppresses a log showing the progress of the estimation.

`rseed(#)` sets the random-number seed. This option can be used to reproduce results for `selection(cv)` and `selection(adaptive)`. The other selection methods, `selection(plugin)`, `selection(bic)`, and `selection(none)`, do not use random numbers. `rseed(#)` is equivalent to typing `set seed #` prior to running `lasso`. See [R] **set seed**.

`grid(`$\#_g$ [`, ratio(#) min(#)`]`)` specifies the set of possible λ's using a logarithmic grid with $\#_g$ grid points.

$\#_g$ is the number of grid points for λ. The default is $\#_g = 100$. The grid is logarithmic with the ith grid point ($i = 1, \ldots, n = \#_g$) given by $\ln \lambda_i = [(i-1)/(n-1)] \ln r + \ln \lambda_{\text{gmax}}$, where $\lambda_{\text{gmax}} = \lambda_1$ is the maximum, $\lambda_{\text{gmin}} = \lambda_n =$ `min(#)` is the minimum, and $r = \lambda_{\text{gmin}}/\lambda_{\text{gmax}} =$ `ratio(#)` is the ratio of the minimum to the maximum.

`ratio(#)` specifies $\lambda_{\text{gmin}}/\lambda_{\text{gmax}}$. The maximum of the grid, λ_{gmax}, is set to the smallest λ for which all the coefficients in the lasso are estimated to be zero (except the coefficients of the *alwaysvars*). λ_{gmin} is then set based on `ratio(#)`. When $p < N$, where p is the total number of *othervars* and *alwaysvars* (not including the constant term) and N is the number of observations, the default value of `ratio(#)` is 1e−4. When $p \geq N$, the default is 1e−2.

`min(#)` sets λ_{gmin}. By default, λ_{gmin} is based on `ratio(#)` and λ_{gmax}, which is computed from the data.

`stop(#)` specifies a tolerance that is the stopping criterion for the λ iterations. The default is 1e−5. This suboption does not apply when the selection method is `selection(plugin)`. Estimation starts with the maximum grid value, λ_{gmax}, and iterates toward the minimum grid value, λ_{gmin}. When the relative difference in the deviance produced by two adjacent λ grid values is less than `stop(#)`, the iteration stops and no smaller λ's are evaluated. The value of λ that meets this tolerance is denoted by λ_{stop}. Typically, this stopping criterion is met before the iteration reaches λ_{gmin}.

Setting `stop(#)` to a larger value means that iterations are stopped earlier at a larger λ_{stop}. To produce coefficient estimates for all values of the λ grid, `stop(0)` can be specified. Note, however, that computations for small λ's can be extremely time consuming. In terms of time, when using `selection(cv)`, `selection(adaptive)`, or `selection(bic)`, the optimal value of `stop(#)` is the largest value that allows estimates for just enough λ's to be computed to identify the minimum of the CV or BIC function. When setting `stop(#)` to larger values, be aware of the consequences of the default λ^* selection procedure given by the default `stopok`. You may want to override the `stopok` behavior by using `strict`.

`cvtolerance(#)` is a rarely used option that changes the tolerance for identifying the minimum CV function. For linear models, a minimum is identified when the CV function rises above a nominal minimum for at least three smaller λ's with a relative difference in the CV function greater than #. For nonlinear models, at least five smaller λ's are required. The default is 1e−3. Setting # to a bigger value makes a stricter criterion for identifying a minimum and brings more assurance that a declared minimum is a true minimum, but it also means that models may need to be fit for additional smaller λ, which can be time consuming. See *Methods and formulas* for [LASSO] **lasso** for more information about this tolerance and the other tolerances.

`bictolerance(#)` is a rarely used option that changes the tolerance for identifying the minimum BIC function. A minimum is identified when the BIC function rises above a nominal minimum for at least two smaller λ's with a relative difference in the BIC function greater than #. The default is 1e−2. Setting # to a bigger value makes a stricter criterion for identifying a minimum and brings more assurance that a declared minimum is a true minimum, but it also means that models may need to be fit for additional smaller λ, which can be time consuming. See *Methods and formulas* in [LASSO] **lasso** for more information about this tolerance and the other tolerances.

`tolerance(#)` is a rarely used option that specifies the convergence tolerance for the coefficients. Convergence is achieved when the relative change in each coefficient is less than this tolerance. The default is `tolerance(1e-7)`.

`dtolerance(#)` is a rarely used option that changes the convergence criterion for the coefficients. When `dtolerance(#)` is specified, the convergence criterion is based on the change in deviance instead of the change in the values of coefficient estimates. Convergence is declared when the relative change in the deviance is less than #. More-accurate coefficient estimates are typically achieved by not specifying this option and instead using the default `tolerance(1e-7)` criterion or specifying a smaller value for `tolerance(#)`.

The following option is available with `lasso` but is not shown in the dialog box:

`penaltywt(`*matname*`)` is a programmer's option for specifying a vector of weights for the coefficients in the penalty term. The contribution of each coefficient to the lasso penalty term is multiplied by its corresponding weight. Weights must be nonnegative. By default, each coefficient's penalty weight is 1.

Remarks and examples

We assume you have read the lasso introduction [LASSO] **Lasso intro**.

Remarks are presented under the following headings:

Lasso fitting and selection methods
selection(cv): Cross-validation
The CV function
Penalized and postselection coefficients
predict
Selecting lambda by hand using lassoselect
More lasso examples

Lasso fitting and selection methods

Lasso finds a vector of coefficient estimates, $\boldsymbol{\beta}$, such that

$$\frac{1}{2N}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}')'(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}')+\lambda\sum_{j=1}^{p}|\beta_j|$$

is minimized for a given value of λ. The first thing to note is that for every λ there is a $\boldsymbol{\beta}$. The second thing is that some of the coefficients, β_j, will be zero. The third thing is that the larger the value of λ, the fewer number of nonzero coefficients there will be.

The goal is to select a λ such that the set of variables corresponding to the nonzero coefficients for that λ has some sort of desirable property. We term the selected λ^*. But remember whenever we talk about the selected λ^*, we are really thinking about the properties of the corresponding set of variables with nonzero coefficients.

Different criteria can be used to select λ^*. The `lasso` command has options for four different selection methods: `selection(cv)`, `selection(adaptive)`, `selection(plugin)`, `selection(bic)`, and `selection(none)`.

selection(cv) comes in two variants: the default, which selects λ^* as the value of λ that minimizes the CV function; and selection(cv, serule), which selects a λ^* that is one standard error from the minimum in the direction of larger λ's (so fewer selected variables than using the minimum in most cases).

selection(adaptive) fits multiple lassos, typically just two, with each lasso using CV. The selected λ^* is the λ selected by the last lasso. See *Adaptive lasso* in [LASSO] **lasso examples**.

selection(plugin) selects λ^* based on an iterative formula. It comes in two variants, the default selection(plugin, heteroskedastic) and selection(plugin, homoskedastic). It is intended to be used as a tool for implementing inferential models. It is not intended to be used for prediction. See [LASSO] **Lasso inference intro**.

selection(bic) selects the λ^* that minimizes the BIC function. Zhang, Li, and Tsai (2010) show that the λ selected by minimizing the BIC will select a set of covariates close to the true set under the conditions described in their article. See *BIC* in [LASSO] **lasso examples**.

selection(none) does not select λ^*. Afterward, you can select λ using the command lassoselect. See *Selecting lambda by hand using lassoselect* below.

We will first explain CV.

selection(cv): Cross-validation

We will illustrate CV using Stata's auto dataset. This is an unrealistic dataset to use with lasso because the number of variables and the number of observations are small. Lasso was invented with the idea of using hundreds or thousands of variables. See [LASSO] **lasso examples** for examples with a large dataset. The small size of the auto dataset, however, is convenient because it does not produce lots of output, and it illustrates some important concepts perfectly.

We load the data.

```
. sysuse auto
(1978 automobile data)
```

We want to model the variable mpg, which is a car's miles per gallon. Choices for type of lasso model are linear, logit, probit, poisson, and cox. Obviously, linear is the only appropriate type of model for mpg. We follow mpg in the command specification with all the other numeric variables in the dataset. foreign and rep78 are categorical variables, so we specify them using factor-variable operator i. to create indicators for their categories. We do not specify the selection() option because selection(cv) is the default.

Before we run `lasso`, we set the random-number seed. CV uses random numbers, so if we want to be able to reproduce our results, we must first set the seed.

```
. set seed 1234
. lasso linear mpg i.foreign i.rep78 headroom weight turn gear_ratio
> price trunk length displacement
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda =  4.69114   no. of nonzero coef. =  0
Folds: 1...5....10    CVF = 33.97832
Grid value 2:     lambda = 4.274392   no. of nonzero coef. =  2
Folds: 1...5....10    CVF = 31.62288
  (output omitted)
Grid value 44:    lambda = .0858825   no. of nonzero coef. = 10
Folds: 1...5....10    CVF = 13.39785
Grid value 45:    lambda = .0782529   no. of nonzero coef. = 11
Folds: 1...5....10    CVF = 13.45168
... cross-validation complete ... minimum found
Lasso linear model                           No. of obs        =         69
                                             No. of covariates =         15
Selection: Cross-validation                  No. of CV folds   =         10

─────────────────────────────────────────────────────────────────────────
         │                                No. of      Out-of-      CV mean
         │                               nonzero       sample   prediction
      ID │     Description      lambda     coef.    R-squared        error
─────────┼───────────────────────────────────────────────────────────────
       1 │    first lambda     4.69114         0      -0.0018     33.97832
      41 │   lambda before    .1135316         8       0.6062      13.3577
    * 42 │ selected lambda    .1034458         8       0.6066      13.3422
      43 │    lambda after    .0942559         9       0.6060     13.36279
      45 │     last lambda    .0782529        11       0.6034     13.45168
─────────────────────────────────────────────────────────────────────────
* lambda selected by cross-validation.
. estimates store autolasso
```

After `lasso` finished, we typed `estimates store autolasso` to keep the results in memory. This lasso was quick to compute, but lassos with lots of observations and lots of variables can take some time to compute, so it is a good idea to store them.

`lasso` almost always produces a long iteration log. In this example, it iterated from grid value 1 with $\lambda = 4.691140$ to grid value 45 with $\lambda = 0.078253$. By default, `selection(cv)` sets up a grid of 100 λ's, which are spaced uniformly on a logarithmic scale. It ended at grid value 45 and did not do any calculations for the 55 smallest λ gird points.

If we look at the output table, we see that the λ at grid value 1 has 0 nonzero coefficients corresponding to it. This is how the first λ is calculated. It is the smallest λ that gives 0 nonzero coefficients. The λ at grid value 100 is set by the `grid()` suboption `ratio(#)`, which specifies the ratio of the last (minimum) λ to the first (maximum) λ. The default for `ratio(#)` in this case is 1e−4.

For each value of λ, coefficients are estimated. The entire list of λ's can be viewed at any time using the postestimation command lassoknots with the option alllambdas. It shows what happened at every iteration.

```
. lassoknots, alllambdas

─────────────────────────────────────────────────────────────────────────────
       │                   No. of   CV mean │
       │                  nonzero     pred. │   Variables (A)dded, (R)emoved,
    ID │     lambda         coef.     error │        or left (U)nchanged
───────┼────────────────────────────────────┼────────────────────────────────
     1 │    4.69114             0  33.97832 │ U
     2 │   4.274392             2  31.62288 │ A weight   length
     3 │   3.894667             2  28.65489 │ U
     4 │   3.548676             2   26.0545 │ U
     5 │   3.233421             2   23.8774 │ U
     6 │   2.946173             2  22.07264 │ U
     7 │   2.684443             2  20.57514 │ U
     8 │   2.445964             2  19.30795 │ U
     9 │   2.228672             2  18.23521 │ U
    10 │   2.030683             2  17.43067 │ U
    11 │   1.850282             2  16.78884 │ U
    12 │   1.685908             2  16.32339 │ U
    13 │   1.536137             2  15.97483 │ U
    14 │   1.399671             2  15.70143 │ U
    15 │   1.275328             3  15.48129 │ A 5.rep78
    16 │   1.162031             3  15.34837 │ U
    17 │     1.0588             3  15.30879 │ U
    18 │   .9647388             3  15.30897 │ U
    19 │   .8790341             4   15.3171 │ A turn
    20 │   .8009431             5  15.32254 │ A gear_ratio
    21 │   .7297895             6  15.31234 │ A price
    22 │    .664957             6  15.28881 │ U
    23 │   .6058841             6  15.26272 │ U
    24 │    .552059             6  15.20981 │ U
    25 │   .5030156             6   15.1442 │ U
    26 │   .4583291             6  15.04271 │ U
    27 │   .4176124             6  14.92838 │ U
    28 │   .3805129             6    14.877 │ U
    29 │   .3467091             6  14.83908 │ U
    30 │   .3159085             7  14.77343 │ A 0.foreign
    31 │    .287844             8  14.67034 │ A 3.rep78
    32 │   .2622728             8  14.53728 │ U
    33 │   .2389732             8  14.35716 │ U
    34 │   .2177434             8  14.15635 │ U
    35 │   .1983997             8  13.95308 │ U
    36 │   .1807744             8  13.77844 │ U
    37 │   .1647149             8  13.62955 │ U
    38 │   .1500821             8    13.519 │ U
    39 │   .1367492             8  13.43867 │ U
    40 │   .1246008             8  13.39141 │ U
    41 │   .1135316             8   13.3577 │ U
  * 42 │   .1034458             8   13.3422 │ U
    43 │   .0942559             9  13.36279 │ A 1.rep78
    44 │   .0858825            10  13.39785 │ A headroom
    45 │   .0782529            11  13.45168 │ A displacement
─────────────────────────────────────────────────────────────────────────────
* lambda selected by cross-validation.
```

As λ gets smaller, there are more nonzero coefficients. As the nonzero coefficients change, variables are added to the model. Sometimes, variables are removed from the model. That is, a coefficient once nonzero becomes zero at a smaller λ. In this example, once added to the model, no variable was ever removed. When there are more potential variables, you will typically see some variables removed as other variables are added.

Usually, the number of nonzero coefficients increases monotonically as λ gets smaller, but not always. Occasionally, the net number of variables in the model goes down, rather than up, in an iteration to a smaller λ.

The λ's at which variables are added or removed are called knots. By default, `lassoknots` shows only the knots—and the λ that minimizes the CV function if it is not a knot. This λ is denoted by λ^* and is indicated in the table with an `*`.

```
. lassoknots

----------------------------------------------------------------------------------
           |                No. of   CV mean |
           |               nonzero     pred. |   Variables (A)dded, (R)emoved,
        ID |    lambda       coef.     error |        or left (U)nchanged
-----------+---------------------------------+------------------------------------
         2 |  4.274392           2  31.62288 | A weight   length
        15 |  1.275328           3  15.48129 | A 5.rep78
        19 |  .8790341           4   15.3171 | A turn
        20 |  .8009431           5  15.32254 | A gear_ratio
        21 |  .7297895           6  15.31234 | A price
        30 |  .3159085           7  14.77343 | A 0.foreign
        31 |   .287844           8  14.67034 | A 3.rep78
      * 42 |  .1034458           8   13.3422 | U
        43 |  .0942559           9  13.36279 | A 1.rep78
        44 |  .0858825          10  13.39785 | A headroom
        45 |  .0782529          11  13.45168 | A displacement
----------------------------------------------------------------------------------
* lambda selected by cross-validation.
```

The CV function

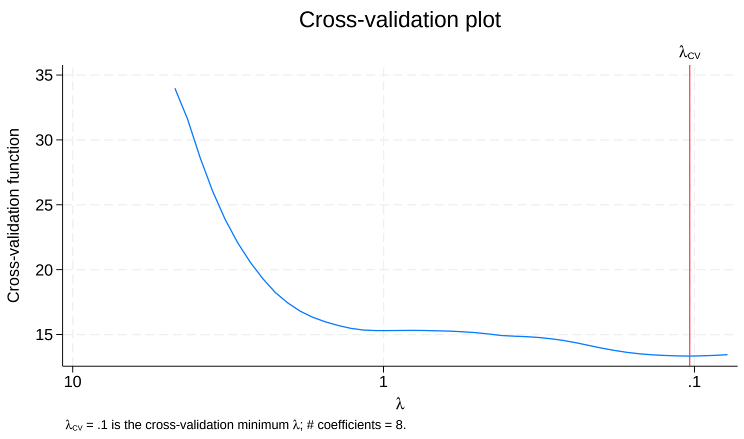
After coefficients are estimated for each λ, the value of the CV function is calculated. CV is done by dividing the data randomly into folds, by default, 10 of them. (This is the step where random numbers are used.)

One fold is chosen, and then a linear regression is fit on the other nine folds using the variables in the model for that λ. Then, with these new coefficient estimates, a prediction is computed for the data of the chosen fold. The mean squared error (MSE) of the prediction is computed. This process is repeated for the other nine folds. The 10 MSEs are then averaged to give the value of the CV function. On the output, the CV function is labeled CV mean prediction error.

By default, `selection(cv)` looks for a minimum of the CV function and then stops once it has found one. We see that models for three λ's past the minimum were fit. For `linear` models, `selection(cv)` needs to see three smaller λ's with larger values of the CV function to declare that it has found a minimum. It sets the selected λ^* to the λ that gave the minimum and stops.

We can plot the CV function using cvplot.

```
. cvplot
```



If we want to see more values of the CV function, we can run lasso again using selection(cv, alllambdas).

```
. set seed 1234

. lasso linear mpg i.foreign i.rep78 headroom weight turn gear_ratio
> price trunk length displacement, selection(cv, alllambdas)

Evaluating up to 100 lambdas in grid ...
Grid value 1:     lambda =  4.69114   no. of nonzero coef. =  0
Grid value 2:     lambda = 4.274392   no. of nonzero coef. =  2
```

(output omitted)

```
Grid value 76:    lambda =  .004375   no. of nonzero coef. = 13
Grid value 77:    lambda = .0039863   no. of nonzero coef. = 13
... change in deviance stopping tolerance reached

10-fold cross-validation with 77 lambdas ...
Fold  1 of 10:  10....20....30....40....50....60....70...
Fold  2 of 10:  10....20....30....40....50....60....70...
```

(output omitted)

```
Fold  9 of 10:  10....20....30....40....50....60....70...
Fold 10 of 10:  10....20....30....40....50....60....70...
... cross-validation complete

Lasso linear model                          No. of obs        =         69
                                            No. of covariates =         15
Selection: Cross-validation                 No. of CV folds   =         10
```

ID	Description	lambda	No. of nonzero coef.	Out-of-sample R-squared	CV mean prediction error
1	first lambda	4.69114	0	-0.0018	33.97832
41	lambda before	.1135316	8	0.6062	13.3577
* 42	selected lambda	.1034458	8	0.6066	13.3422
43	lambda after	.0942559	9	0.6060	13.36279
77	last lambda	.0039863	13	0.5765	14.36306

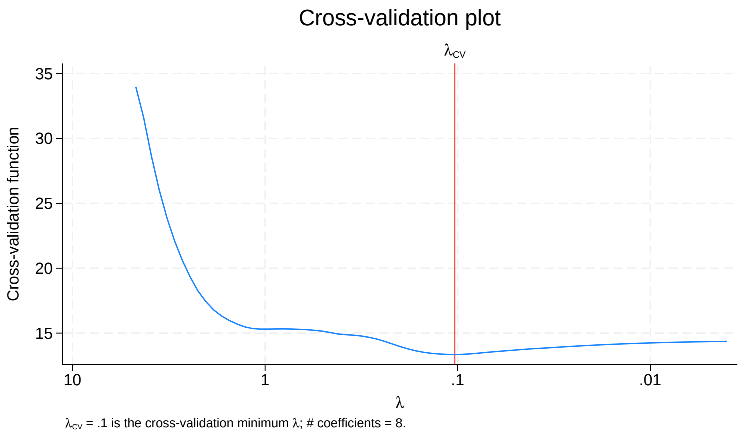
```
* lambda selected by cross-validation.
```

The iteration log is in a different order than it was earlier. Here we see messages about all the grid values first and then the folds of the CV. Earlier, we saw grid values and then folds, and then grid values and then folds, etc. With `alllambdas`, coefficient vectors for all the λ's are estimated first, and then CV is done. When we are not going to stop when a minimum is found, this is a slightly faster way of doing the computation.

The selected λ^* and the values of the CV function and R^2 are exactly the same—if we set the random-number seed to the same value we used before. Had we forgotten to set the random-number seed or set it to a different value, the values of the CV function and R^2 would be slightly different, and frequently, even the selected λ^* is different.

Let's plot the CV function again with these additional λ's.

```
. cvplot
```



The suboption `alllambdas` lied to us. It did not give us all λ's. There are 100 λ's in the grid. It showed us 77 of them this time, not all 100.

There is another rule that determines when the iteration over λ's ends. It is the stopping tolerance set by the option `stop(#)`. When the deviance calculated from the estimated coefficients changes little from one λ to the next, the iteration stops. The idea behind this stopping rule is that it means the CV function would flatten out at this point, and there is no reason to continue estimating coefficients for smaller λ's. If you really want to see the smallest λ, specify `stop(0)` like so:

```
. lasso linear ..., selection(cv, alllambdas) stop(0)
```

Note that `stop(#)` is not specified as a suboption of the `selection(cv)` option. The `stop(#)` stopping rule has nothing to do with CV. It is based solely on the change in deviance produced by the coefficient estimates.

Why do we have all these rules for ending the iteration over λ as soon as possible? The reason is because the smaller the λ, the longer the computation time. If you have lots of observations and lots of variables, you still see the iteration log going slower and slower with each successive λ. There is no point in burning lots of computer time—except if you want to draw a prettier picture of the CV function.

Advanced note: If you want more evidence that the identified minimum is the true minimum, you are better off setting the option `cvtolerance(#)` to a larger value than specifying `alllambdas`. You will get assurance in much shorter time.

Another advanced note: Setting `stop(0)` without specifying `alllambdas` is sometimes useful. See [LASSO] **lasso fitting** for details.

Penalized and postselection coefficients

We have discussed how lasso fitting and CV works without even mentioning the purpose of lasso. But you read [LASSO] **Lasso intro**, right? The purposes of lasso are covered there. We are assuming here that our purpose for this lasso is to build a predictive model for `mpg`.

To get predictions after `lasso`, we use `predict`, just as we use `predict` after `regress`. But we have two choices after `lasso`. After `lasso`, we can use penalized coefficients to compute our predictions, or we can use postselection coefficients.

Actually, there are three types of coefficients after `lasso`. What we refer to as `standardized`, `penalized`, and `postselection`.

Before we minimize the objective function

$$\frac{1}{2N}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}')'(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}') + \lambda\sum_{j=1}^{p}|\beta_j|$$

we standardize the columns of $\mathbf{X}$ (that is, the potential variables in the model) so that they each have mean 0 and standard deviation 1. Otherwise, the term

$$\sum_{j=1}^{p}|\beta_j|$$

would be dependent on the scales of the variables.

`standardized` refers to the coefficients of the standardized variables exactly as estimated by the minimization of the objective function.

When we are doing lasso for prediction, we are not supposed to care about the values of the coefficients or look at them. (Read [LASSO] **Lasso intro**!) However, even we could not follow our own advice, so we developed a command, `lassocoef`, especially for listing the coefficients.

Let's list the coefficients of the standardized variables.

```
. lassocoef, display(coef, standardized)

---------------------------
             |     active
-------------+-------------
   0.foreign |    1.49568
             |
       rep78 |
          3  |  -.3292316
          5  |   1.293645
             |
      weight |  -.2804677
        turn |  -.7378134
  gear_ratio |   1.378287
       price |  -.2809065
      length |  -2.942432
       _cons |          0
---------------------------
Legend:
  b - base level
  e - empty cell
  o - omitted
```

The coefficients of the standardized variables seem to be the same order of magnitude as we expect.

penalized refers to the coefficients from the minimization of the objective function with the standardization unwound. standardized, strictly speaking, gives the penalized coefficients of the standardized variables. penalized gives the penalized coefficients of the unstandardized variables. Let's list them.

```
. lassocoef, display(coef, penalized)

---------------------------
             |     active
-------------+-------------
   0.foreign |   3.250554
             |
       rep78 |
          3  |  -.6641369
          5  |   3.533896
             |
      weight |  -.0003563
        turn |   -.167352
  gear_ratio |   3.000733
       price |  -.0000972
      length |  -.1303001
       _cons |   42.62583
---------------------------
Legend:
  b - base level
  e - empty cell
  o - omitted
```

The third type, `postselection`, is computed by taking the selected variables, estimating a linear regression with them, and using those coefficients.

```
. lassocoef, display(coef, postselection)

------------------------
             |    active
-------------+----------
   0.foreign |  4.769344
             |
       rep78 |
          3  | -1.010493
          5  |  4.037817
             |
      weight |  -.000157
        turn | -.2159788
  gear_ratio |  3.973684
       price | -.0000582
      length | -.1355416
       _cons |  40.79938
------------------------
Legend:
  b - base level
  e - empty cell
  o - omitted
```

We can duplicate these results with `regress`.

```
. regress mpg 0bn.foreign 3bn.rep78 5bn.rep78 weight turn gear_ratio
> price length

      Source |       SS           df       MS      Number of obs   =        69
-------------+----------------------------------   F(8, 60)        =     22.14
       Model |  1748.04019         8  218.505024   Prob > F        =    0.0000
    Residual |  592.162704        60  9.86937839   R-squared       =    0.7470
-------------+----------------------------------   Adj R-squared   =    0.7132
       Total |   2340.2029        68  34.4147485   Root MSE        =    3.1416

------------------------------------------------------------------------------
         mpg | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
     foreign |
   Domestic  |   4.769344   1.596469     2.99   0.004     1.575931    7.962757
             |
       rep78 |
          3  |  -1.010493   .8775783    -1.15   0.254    -2.765911    .7449251
          5  |   4.037817   1.262631     3.20   0.002     1.512178    6.563455
             |
      weight |   -.000157   .0021651    -0.07   0.942    -.0044878    .0041739
        turn |  -.2159788   .1886946    -1.14   0.257    -.5934242    .1614665
  gear_ratio |   3.973684   1.603916     2.48   0.016     .7653732    7.181994
       price |  -.0000582   .0001996    -0.29   0.772    -.0004574    .0003411
      length |  -.1355416   .0595304    -2.28   0.026    -.2546201   -.0164632
       _cons |   40.79938   9.206714     4.43   0.000     22.38321    59.21555
------------------------------------------------------------------------------
```

What are you doing looking at the p-values! If we are not supposed to look at the coefficients, surely this applies many times over to p-values. We looked, too. And we see that the lasso selected a bunch with big p-values. Lasso does not care about p-values. Its sole goal is to build a model that is good for prediction, and it thought these variables would help do that. Maybe it is just fitting random noise, and CV as a selection method is known to do that. Adding extra variables that are fitting only random noise is called “overselecting”.

We want to point out that although `rep78` has five categories, lasso selected only two of them, `rep78` = 3 and `rep78` = 5, to be in the final model. See *Factor variables in lasso* in [LASSO] **lasso examples** and [LASSO] **Collinear covariates**.

predict

The options `penalized` and `postselection` carry over to `predict`. We can

```
    predict yhat, penalized
```

Or we can

```
    predict yhat, postselection
```

If we simply type

```
    predict yhat
```

we get penalized.

For linear models, postselection coefficients give predictions that are theoretically slightly better than those given by penalized coefficients. In practice, however, the difference in the prediction is small.

For logit, probit, Poisson, and Cox models, there is no theory for the postselection predictions. Only the penalized predictions have a theoretical basis. So the default, `penalized`, is recommended for these models.

See [LASSO] **lasso postestimation**.

Selecting lambda by hand using lassoselect

We can change the selected λ^* if we want. It is easy to do. Recall that we stored our original lasso results in memory using

```
    . estimates store name
```

We can then compare these results with those from other lassos. We show examples of this in [LASSO] **lasso examples**. Note, however, that `estimates store` only saves them in memory. To save the results to disk, use

```
    . estimates save filename
```

See [LASSO] **estimates store**.

We restore our previous results.

```
    . estimates restore autolasso
    (results autolasso are active now)
```

Let’s run `lassoknots` again with options to show R^2. There are two types of R^2 available. See [LASSO] **lassoknots** for a discussion. The one labeled out-of-sample is the better one to look at.

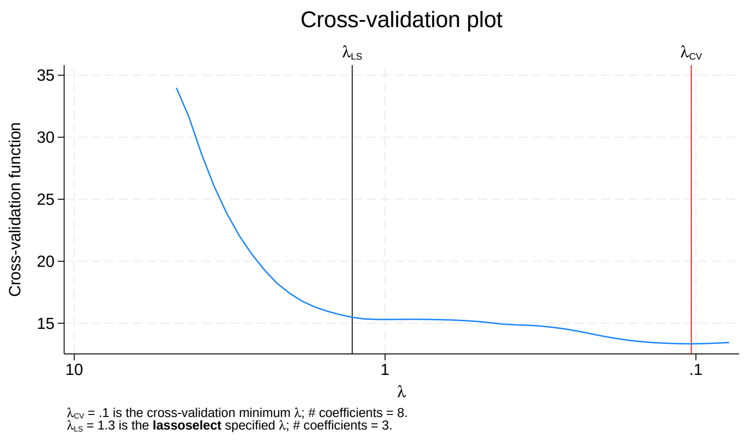
```
. lassoknots, display(cvmpe r2 osr2)

─────────────────────────────────────────────────────────────────────
         │                   CV mean      Out-of-
         │                     pred.       sample   In-sample
      ID │      lambda         error    R-squared   R-squared
─────────┼───────────────────────────────────────────────────────────
       2 │   4.274392     31.62288       0.0676      0.1116
      15 │   1.275328     15.48129       0.5435      0.6194
      19 │   .8790341      15.3171       0.5484      0.6567
      20 │   .8009431     15.32254       0.5482      0.6627
      21 │   .7297895     15.31234       0.5485      0.6684
      30 │   .3159085     14.77343       0.5644      0.7030
      31 │    .287844     14.67034       0.5675      0.7100
    * 42 │   .1034458      13.3422       0.6066      0.7422
      43 │   .0942559     13.36279       0.6060      0.7431
      44 │   .0858825     13.39785       0.6050      0.7439
      45 │   .0782529     13.45168       0.6034      0.7449
─────────────────────────────────────────────────────────────────────
* lambda selected by cross-validation.
```

That λ with ID $=$ 15 looks almost as good as the one CV picked. Let’s select it.

```
. lassoselect id = 15
ID = 15  lambda = 1.275328 selected
```

The new selected λ^* is shown on `cvplot`.

```
. cvplot
```



We can look at the coefficients and compare them with the earlier results.

```
. lassocoef autolasso ., display(coef, postselection)

─────────────────────────────────────
             │  autolasso     active
─────────────┼───────────────────────
   0.foreign │   4.769344
             │
       rep78 │
          3  │  -1.010493
          5  │   4.037817   2.782347
             │
      weight │   -.000157  -.0024045
        turn │  -.2159788
  gear_ratio │   3.973684
       price │  -.0000582
      length │  -.1355416  -.1120782
       _cons │   40.79938   49.23984
─────────────────────────────────────
Legend:
  b - base level
  e - empty cell
  o - omitted
```

The earlier lasso was stored as `autolasso`. When we use `lassoselect`, it is just like running a new lasso. New estimation results are created. The period (`.`) used as an argument to `lassocoef` means the current estimation results. If we want to compare these results with others in the future, we can use `estimates store` and store them under a new name. Then we can use this name with `lassocoef`.

Our new selected λ^* certainly gives a more parsimonious model. Too bad we do not have any theoretical basis for choosing it.

More lasso examples

We have yet to give examples for many important features. They include using split samples to evaluate predictions, fitting logit, probit, Poisson, and Cox models, and selecting λ^* using adaptive lasso.

In [LASSO] **lasso examples**, we illustrate these capabilities using a dataset with lots of variables. We also show how to use the `vl` commands, a system for managing large variable lists.

Stored results

`lasso` stores the following in `e()`:

Scalars

`e(N)`	number of observations
`e(N_clust)`	number of clusters
`e(k_allvars)`	number of potential variables
`e(k_nonzero_sel)`	number of nonzero coefficients for selected model
`e(k_nonzero_cv)`	number of nonzero coefficients at CV mean function minimum
`e(k_nonzero_serule)`	number of nonzero coefficients for one-standard-error rule
`e(k_nonzero_min)`	minimum number of nonzero coefficients among estimated λ's
`e(k_nonzero_max)`	maximum number of nonzero coefficients among estimated λ's
`e(k_nonzero_bic)`	number of nonzero coefficients at BIC function minimum
`e(lambda_sel)`	value of selected λ^*
`e(lambda_gmin)`	value of λ at grid minimum
`e(lambda_gmax)`	value of λ at grid maximum

e(lambda_last)	value of last λ computed
e(lambda_cv)	value of λ at CV mean function minimum
e(lambda_serule)	value of λ for one-standard-error rule
e(lambda_bic)	value of λ at BIC function minimum
e(ID_sel)	ID of selected λ^*
e(ID_cv)	ID of λ at CV mean function minimum
e(ID_serule)	ID of λ for one-standard-error rule
e(ID_bic)	ID of λ at BIC function minimum
e(cvm_min)	minimum CV mean function value
e(cvm_serule)	CV mean function value at one-standard-error rule
e(devratio_min)	minimum deviance ratio
e(devratio_max)	maximum deviance ratio
e(L1_min)	minimum value of ℓ_1-norm of penalized unstandardized coefficients
e(L1_max)	maximum value of ℓ_1-norm of penalized unstandardized coefficients
e(L2_min)	minimum value of ℓ_2-norm of penalized unstandardized coefficients
e(L2_max)	maximum value of ℓ_2-norm of penalized unstandardized coefficients
e(ll_sel)	log-likelihood value of selected model
e(n_lambda)	number of λ's
e(n_fold)	number of CV folds
e(stop)	stopping rule tolerance
Macros	
e(cmd)	lasso
e(cmdline)	command as typed
e(depvar)	name of dependent variable
e(allvars)	names of all potential variables
e(allvars_sel)	names of all selected variables
e(alwaysvars)	names of always-included variables
e(othervars_sel)	names of other selected variables
e(post_sel_vars)	all variables needed for postlasso
e(clustvar)	name of cluster variable
e(lasso_selection)	selection method
e(sel_criterion)	criterion used to select λ^*
e(plugin_type)	type of plugin λ
e(model)	linear, logit, probit, poisson, or cox
e(title)	title in estimation output
e(rngstate)	random-number state used
e(properties)	b
e(predict)	program used to implement predict
e(marginsnotok)	predictions disallowed by margins
Matrices	
e(b)	penalized unstandardized coefficient vector
e(b_standardized)	penalized standardized coefficient vector
e(b_postselection)	postselection coefficient vector
Functions	
e(sample)	marks estimation sample

In addition to the above, the following is stored in r():

Matrices	
r(table)	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in r() are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

This section provides the methods and formulas for `lasso` and `elasticnet`.

Methods and formulas are presented under the following headings:

Lasso and elastic-net objective functions

`lasso` and `elasticnet` estimate the parameters by finding the minimum of a penalized objective function.

The penalized objective function of the lasso for the `linear`, `logit`, `probit`, or `poisson` model is

$$Q_L = \sum_{i=1}^{N} \widetilde{w}_i f(y_i, \beta_0 + \mathbf{x}_i\boldsymbol{\beta}') + \lambda \sum_{j=1}^{p} \kappa_j |\beta_j| \tag{1}$$

where N is the number of observations, $\widetilde{w}_i$ is the normalized observation-level weight, $f(\cdot)$ is the likelihood contribution for the `regress`, `logit`, `probit`, or `poisson` model, β_0 is the intercept, $\mathbf{x}_i$ is the $1 \times p$ vector of covariates, $\boldsymbol{\beta}$ is the $1 \times p$ vector of coefficients, λ is the lasso penalty parameter, which must be greater than or equal to 0, and κ_j are coefficient-level weights (which by default are all 1).

The normalized weights $\widetilde{w}_i$ sum to 1. That is,

$$\widetilde{w}_i = \frac{w_i}{\sum_{i=1}^{N} w_i}$$

where w_i is the original observation-level weight. If weights are not specified with `lasso`, $w_i = 1$ and $\widetilde{w}_i = 1/N$.

When the model is `linear`,

$$f(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}) = \frac{1}{2}(y_i - \beta_0 - \mathbf{x}_i\boldsymbol{\beta}')^2$$

When the model is `logit`,

$$f(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}) = -y_i(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}') + \ln\{1 + \exp(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')\}$$

When the model is `probit`,

$$f(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}) = -y_i \ln\{\Phi(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')\} - (1 - y_i)\ln\{1 - \Phi(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')\}$$

When the model is `poisson`,

$$f(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}) = -y_i(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}') + \exp(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')$$

The penalized objective function of the lasso for the cox model is

$$Q_L = -\sum_{j=1}^{N_f} \sum_{i \in D_j} \widetilde{w}_i \left[\mathbf{x}_i \boldsymbol{\beta}' - \ln \left\{ \sum_{\ell \in R_j} \widetilde{w}_\ell \exp(\mathbf{x}_\ell \boldsymbol{\beta}') \right\} \right] + \lambda \sum_{j=1}^{p} \kappa_j |\beta_j|$$

where j indexes the ordered failure times $t_{(j)}$, $j = 1, \dots, N_f$; D_j is the set of observations that fail at $t_{(j)}$; and R_j is the set of observations k that are at risk at time $t_{(j)}$ (that is, all k such that $t_{0k} < t_{(j)} \leq t_k$, and t_{0k} is the entry time for the kth observation). The first term in Q_L is the weighted negative partial log-likelihood function of the Cox proportional hazards model. There is no constant term β_0 because the constant term is absorbed in the baseline hazard function.

Ties are handled using the Breslow approximation (Breslow 1974). The other methods of handling ties that are options for stcox—the Efron method, the exact marginal-likelihood method, and the exact partial-likelihood method—are not available with lasso cox.

The penalized objective function of elastic net for the linear, logit, probit, and poisson models is

$$Q_{\text{en}} = \sum_{i=1}^{N} \widetilde{w}_i f(y_i, \beta_0 + \mathbf{x}_i \boldsymbol{\beta}') + \lambda \sum_{j=1}^{p} \kappa_j \left\{ \frac{1-\alpha}{2} \beta_j^2 + \alpha |\beta_j| \right\} \tag{2}$$

where α is the elastic-net penalty parameter and α can take on values only in $[0, 1]$.

The penalized objective function of elastic net for the cox model is

$$Q_{\text{en}} = -\sum_{j=1}^{N_f} \sum_{i \in D_j} \widetilde{w}_i \left[\mathbf{x}_i \boldsymbol{\beta}' - \ln \left\{ \sum_{\ell \in R_j} \widetilde{w}_\ell \exp(\mathbf{x}_\ell \boldsymbol{\beta}') \right\} \right] + \lambda \sum_{j=1}^{p} \kappa_j \left\{ \frac{1-\alpha}{2} \beta_j^2 + \alpha |\beta_j| \right\}$$

Some values for α and λ cause elastic net to reduce to the objective function of another estimator of interest. There are three special cases to note:

1. Lasso is a special case of elastic net. When $\alpha = 1$, the objective function in (2) reduces to the lasso objective function in (1).
2. Ridge regression is a special case of the elastic net. When $\alpha = 0$ and $\lambda > 0$, (2) reduces to the objective function for the ridge-regression estimator.
3. When $\lambda = 0$ in (2), there is no penalty term, and Q_{en} reduces to the objective function for the unpenalized maximum-likelihood estimator.

When $0 < \alpha < 1$ and $\lambda > 0$, (2) is the objective function for an elastic-net estimator that does not reduce to a special case.

We discuss methods that apply to the lasso estimator and to the elastic-net estimator in this section because the same algorithm is used to estimate the coefficients. We discuss the optimization procedure in terms of the elastic-net objective function Q_{en} because it reduces to the lasso estimator when $\alpha = 1$.

We discuss the methods for ridge regression in *Methods and formulas* in [LASSO] **elasticnet** because a different algorithm is used to estimate the coefficients.

By default, the coefficient-level weights are 1 in (1) and (2). They may be specified using the option penaltywt(). If the cluster() option is specified, the log likelihood is computed as the sum of log likelihood at the cluster levels. This option is not allowed for the cox model.

The penalized objective function of the lasso with cluster is

$$Q_L = \sum_{i=1}^{N_{\text{clust}}} \left\{ \sum_{t=1}^{T_i} \widetilde{\widetilde{w}}_{it} f(y_{it}, \beta_0 + \mathbf{x}_{it} \boldsymbol{\beta}') \right\} + \lambda \sum_{j=1}^{p} \kappa_j |\beta_j|$$

where N_{clust} is the total number of clusters and T_i is the number of observations in cluster i. For the tth observation in cluster i, $\widetilde{\widetilde{w}}_{it}$ is its normalized observational level weight, y_{it} is the dependent variable, and $\mathbf{x}_{it}$ are the covariates.

The normalized weights $\widetilde{\widetilde{w}}_{it}$ are defined as

$$\widetilde{\widetilde{w}}_{it} = \frac{\widehat{w}_{it}}{\sum_{i=1}^{N_{\text{clust}}} \sum_{t=1}^{T_i} \widehat{w}_{it}}$$

where $\widehat{w}_{it}$ are the cluster-level normalized weights. For `fweights`, $\widehat{w}_{it} = w_{it} / \sum_{t=1}^{T_i} w_{it}$. For `iweights`, $\widehat{w}_{it} = w_{it}/T_i$.

The penalized objective function of elastic net with cluster is

$$Q_{\text{en}} = \sum_{i=1}^{N_{\text{clust}}} \left\{ \sum_{t=1}^{T_i} \widetilde{\widetilde{w}}_{it} f(y_{it}, \beta_0 + \mathbf{x}_{it}\boldsymbol{\beta}') \right\} + \lambda \sum_{j=1}^{p} \kappa_j \left\{ \frac{1-\alpha}{2} \beta_j^2 + \alpha |\beta_j| \right\}$$

Coordinate descent

`lasso` and `elasticnet` use the coordinate descent algorithm to minimize Q_{en} for given values of λ and α.

The coordinate descent algorithm for lasso problems was first applied to lasso as a "shooting algorithm" in Fu (1998). Daubechies, Defrise, and Mol (2004) also discussed using coordinate descent for lasso. The combination of Friedman et al. (2007), Friedman, Hastie, and Tibshirani (2010), and Hastie, Tibshirani, and Wainwright (2015) provide a complete introduction to using the coordinate descent algorithm for lasso and elastic net, and these references detail the formulas implemented in `lasso` and `elasticnet`.

The numerical problem is made much easier and more stable by standardizing all the covariates to have mean 0 and standard deviation 1. The standardization also removes β_0 from the problem when the model is `regress`.

Minimization problems are solved by finding the parameter values that set the first derivative equations to 0. The first derivative equations are known as score equations in statistics. When the score equations for all the elements in $\boldsymbol{\beta}$ are well defined, we frequently use a version of Newton's method that produces a series of updated guesses for $\boldsymbol{\beta}$ that get increasingly close to solving the score equations. When the updated guess is close enough to solving the score equations, the algorithm converges and we have our estimates.

Unfortunately, Q_{en} is not always differentiable. When $\lambda > 0$ and the kth element in $\boldsymbol{\beta}$ is 0, Q_{en} is not differentiable. Convex analysis provides a way of getting a generalized score equation for the kth element of $\boldsymbol{\beta}$ that handles the case in which $\boldsymbol{\beta}_k$ is 0. It is not feasible to write down equations for all p generalized score equations at the same time. It is too complicated.

In general terms, coordinate descent is a solve-and-replace algorithm that repeatedly solves each generalized score equation for a new coefficient value until a minimum of Q_{en} is found. For those familiar with the Gauss–Seidel algorithm, coordinate descent is basically Gauss–Seidel on the generalized score equations. Quandt (1984) discusses the Gauss–Seidel algorithm.

To be more specific, we provide an outline of the implemented coordinate descent algorithm.

Step 1: Specify initial values.

a. Set each coefficient to an initial value $\widehat{\boldsymbol{\beta}}_k = v_k$. We refer to $\widehat{\boldsymbol{\beta}}$ as the current coefficient vector.

b. Initialize each coefficient in the previous coefficient vector $\widetilde{\boldsymbol{\beta}}$ to be a missing value.

c. Initialize the difference, Δ, between the current and the previous coefficient vectors to be a missing value.

Step 2: As long as Δ is larger than `tolerance(#)`, do the following.

a. Set each coefficient in the current coefficient vector to the value that sets its generalized score equation to 0. In other words, set

$$\widehat{\boldsymbol{\beta}}_k = g_k(y, \mathbf{x}, \widehat{\beta}_1, \dots \widehat{\beta}_{k-1}, \widehat{\beta}_{k+1}, \dots \widehat{\beta}_p)$$

where $g_k(y, \mathbf{x}, \widehat{\beta}_1, \dots \widehat{\beta}_{k-1}, \widehat{\beta}_{k+1}, \dots \widehat{\beta}_p)$ is the expression for $\widehat{\beta}_k$ that sets the generalized score equation with respect to $\widehat{\beta}_k$ to 0.

b. Let Δ be the largest of the relative differences between $\widehat{\boldsymbol{\beta}}$ and $\widetilde{\boldsymbol{\beta}}$.

c. Set $\widetilde{\boldsymbol{\beta}} = \widehat{\boldsymbol{\beta}}$.

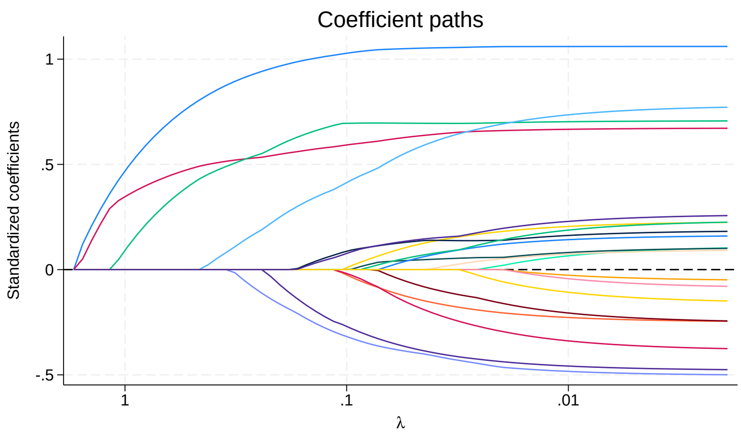
The algorithm converges when step 2 finishes and $\widehat{\boldsymbol{\beta}}$ contains the values that minimize Q_{en} for given values of λ and α.

When the model is `regress`, Hastie, Tibshirani, and Wainwright (2015, eq. 4.4) provide a formula for $g_k(y, \mathbf{x}, \widehat{\beta}_1, \dots \widehat{\beta}_{k-1}, \widehat{\beta}_{k+1}, \dots \widehat{\beta}_p)$. This coordinate descent algorithm is discussed in Hastie, Tibshirani, and Wainwright (2015, chap. 4 and 5).

When the model is `logit`, `probit`, `poisson`, or `cox` the objective function can be minimized by extensions to the method of iteratively reweighted least squares discussed by Nelder and Wedderburn (1972). See Hastie, Tibshirani, and Wainwright (2015, chap. 3) and Friedman, Hastie, and Tibshirani (2010) for details.

Grid of values for λ

For any given value of $0 < \alpha \leq 1$, letting λ decrease from ∞ to 0 creates a vector of coefficient paths. When λ is large enough, all the coefficients are 0. Holding α fixed and decreasing λ from a large value to 0 induces coefficient paths in which each coefficient emerges from 0. In a particular lasso example, we see the following:



In this example, there are fewer covariates than observations, so at $\lambda = 0$, each coefficient path has the value of its unpenalized estimate.

The convention that has emerged following Hastie, Tibshirani, and Wainwright (2015) is to consider a few candidate values for α and a grid of 100 or so candidate values for λ. The default number of grid points is 100, and it can be changed by specifying option `grid(#)`. The candidate values for α are specified by option `alpha()` in `elasticnet`.

The largest value in the grid is the smallest value for which all the coefficients are zero, and we denote it by λ_{gmax}. The smallest value in the grid is λ_{gmin}, where $\lambda_{\text{gmin}} = r\lambda_{\text{gmax}}$ and r is set by the option `grid(, ratio(#))`. The grid is logarithmic with the ith grid point given by $\ln \lambda_i = [(i-1)/(n-1)] \ln r + \ln \lambda_{\text{gmax}}$, where n is the number of grid points.

How to choose the penalty parameter

To use a lasso, we need to decide which value of λ is best. We denote the selected λ as λ^*.

Some methods for choosing λ^* are designed or advertised as facilitating the ability of the lasso as a covariate selection technique. Some authors seem to advocate using the covariates selected by lasso as if this estimate always picked out the true covariates. Unfortunately, the lasso estimate of which covariates to include is too noisy to be treated as without error in subsequent steps, unless all the not-zero coefficients are sufficiently large. This “beta-min” condition is widely viewed as too strong for applied work. See Leeb and Pötscher (2008), Belloni and Chernozhukov (2011), Belloni, Chernozhukov, and Hansen (2014a), and Chernozhukov et al. (2018) for discussions that have led to the rejection of beta-min assumptions. See *Remarks and examples* in [LASSO] **Lasso inference intro** for an introduction to commands that produce reliable inference without a beta-min condition.

The four methods for selecting λ^* for lasso are CV, adaptive lasso, plugin estimators, and BIC.

CV finds the λ^* that will produce coefficient estimates that predict best out of sample. When λ^* is selected by CV and the nonzero coefficients are used for covariate selection, the process tends to select some covariates that do not belong in the model—in addition to ones that belong. See Bühlmann and van de Geer (2011, sec. 2.5.1) for a discussion and further references. This is due to its larger bound on the number of covariates it will find. See Chetverikov, Liao, and Chernozhukov (2019) and their sparsity-bound results.

Adaptive lasso was derived by Zou (2006) and refined by Bühlmann and van de Geer (2011) to provide more reliable covariate selection. As mentioned above, it will not provide mistake-free covariate selection without the widely rejected "beta-min" condition. See section 7.8.6 of Bühlmann and van de Geer (2011) for a discussion of the versions of the beta-min and section 7.10 for a frank conclusion on the difficulty of the problem. While it is not mistake free, covariate selection based on the adaptive lasso will produce a more parsimonious model than covariate selection based on a λ^* selected by CV.

The plugin estimators were designed to pick λ^* to produce accurate results for the subsequently estimated lasso coefficients. See Bickel, Ritov, and Tsybakov (2009), Belloni and Chernozhukov (2011), Belloni, Chernozhukov, and Hansen (2014a, 2014b). These estimators for λ^* are primarily used as part of estimation methods that are robust to the covariate selection mistakes a lasso makes with any choice of λ^*. Plugin estimators for λ^* select a more parsimonious model than does CV. Simulations indicate that plugin-based lassos select fewer covariates than adaptive lasso when there are small coefficients in the true model, but there are no formal results.

BIC selects the λ^* that will produce coefficient estimates that minimize the BIC. Our simulations show that BIC avoids the overselection problem seen in CV and is often faster. BIC tends to select models similar to those of the plugin method but can be applied to a more general class of models.

CV is implemented for `lasso`, `elasticnet`, and `sqrtlasso`. Adaptive lasso is implemented for `lasso`. Plugin estimators are implemented for `lasso` and for `sqrtlasso`. BIC is implemented for `lasso`, `elasticnet`, and `sqrtlasso`.

How CV is performed

CV finds the model that minimizes an out-of-sample prediction error, also known as the CV function. We denote the CV function for the model with parameters θ by $\text{CV}(\theta)$. Formally, CV finds

$$\widehat{\theta} = \arg\ \min_{\theta\in\Theta}\{\text{CV}(\theta)\}$$

For `lasso` or `sqrtlasso`, Θ is the set of λ grid values. For `elasticnet`, Θ is the set of all pairs (λ, α), where λ is in the λ grid and α is one of the specified candidate values.

The value of $\text{CV}(\theta)$ for each $\theta \in \Theta$ is stored in the estimation results after CV is performed. This allows postestimation commands like `cvplot` to plot or display values of the CV function for ranges of θ values.

Here is how $\text{CV}(\theta)$ is computed.

1. Randomly partition the data into K folds.
2. Do the following for each fold $k \in \{1, \ldots, K\}$.
   a. Estimate the parameters of the model for specified θ using the observations not in fold k.
   b. Use the estimates computed in step 2a to fill in the out-of-sample deviance for the observations in fold k.
3. For each model θ, compute the mean of the out-of-sample deviance.
4. The value of $\theta \in \Theta$ with the smallest mean out-of-sample deviance minimizes the CV function.

For the `cox` model, we use the approach in van Houwelingen et al. (2006) to compute the deviance in step 2b. Especially,

$$\widehat{\text{Dev}}_{\lambda}^{k} = \text{Dev}\{\widehat{\theta}^{-k}(\lambda)\} - \text{Dev}^{-k}\{\widehat{\theta}^{-k}(\lambda)\}$$

where $\widehat{\theta}^{-k}(\lambda)$ are the estimates obtained in step 2a, $\text{Dev}\{\widehat{\theta}^{-k}(\lambda)\}$ is the deviance using the full sample and $\widehat{\theta}^{-k}(\lambda)$, and $\text{Dev}^{-k}\{\widehat{\theta}^{-k}(\lambda)\}$ is the deviance using the observations not in the kth fold and $\widehat{\theta}^{-k}(\lambda)$.

For the details of deviance, see *Methods and formulas* in [LASSO] **lassogof**.

Adaptive lasso

Adaptive lasso is a sequence of CV lassos, each at least as parsimonious as the previous one. Mechanically, adaptive lasso is implemented in the following way.

Step A:

Get the initial coefficient estimates and denote them $\widehat{\boldsymbol{\beta}}$. By default, these estimates come from a cross-validated lasso. Optionally, they come from an unpenalized model or from a ridge estimator with λ selected by CV. Zou (2006, 1423) recommends ridge when collinearity is a problem.

Step B:

a. Exclude covariates for which $\widehat{\beta}_j = 0$.

b. Construct coefficient level weights for included covariates, $\kappa_j = 1/|\widehat{\beta}_j|^{\delta}$, where δ is the power to which the weight is raised. By default, $\delta = 1$. To specify another value for δ, use option `selection(adaptive, power(#))`.

Each adaptive step selects either the covariates selected by the previous step or a proper subset of them.

The option `selection(adaptive, step(#))` counts all lassos performed. So the default # = 2 means one adaptive step is done.

Plugin estimators

Heuristically, we get good lasso coefficient estimates when λ^* is large enough to dominate the noise that is inherent in estimating the coefficients when the penalty-loadings κ_j are at their optimal levels. Plugin estimators use the structure of the model and advanced theoretical results to find the smallest λ that dominates the noise, given estimates of the penalty loadings.

For simplicity and compatibility with the rest of the documentation, we did not divide λ by N in (1). Multiply our formulas for λ by N to compare them with those in the cited literature.

As discussed by Bickel, Ritov, and Tsybakov (2009), Belloni and Chernozhukov (2011), Belloni et al. (2012), and Belloni, Chernozhukov, and Wei (2016), the estimation noise is a function of the largest of the absolute values of the score equations of the unpenalized estimator. In particular, when the penalty loadings κ_j are at optimal values, λ^* is chosen so that

$$P\left(\lambda^* \geq c\,\text{max}_{1\leq j\leq p}\left|\frac{1}{N\kappa_j}\sum_{i=1}^{N}\mathbf{h}_j(y_i,\mathbf{x}_i\boldsymbol{\beta}_0')\right|\right) \rightarrow_N 1$$

where c is a constant, and

$$\frac{1}{N}\sum_{i=1}^{N}\mathbf{h}_j(y_i,\mathbf{x}_i\boldsymbol{\beta}_0')$$

is the jth score from the unpenalized estimator at the true coefficients $\boldsymbol{\beta}_0$. The optimal values of the penalty loadings normalize the scores of the unpenalized estimator to have unit variance.

Belloni and Chernozhukov (2011), Belloni et al. (2012), and Belloni, Chernozhukov, and Wei (2016) derive values for λ^* and estimators for κ_j for a variety of models. This firm theoretical structure keeps the lasso with a plugin estimator from including too many irrelevant covariates and provides it with a fast rate of convergence.

In all the implemented methods described below, we use the following notation:

$c = 1.1$ per the recommendation of Belloni and Chernozhukov (2011);

N is the sample size;

$\gamma = 0.1/\ln[\max\{p, N\}]$ is the probability of not removing a variable when it has a coefficient of zero;

p is the number of candidate covariates in the model.

Two plugin estimators are implemented for `lasso linear`:

- `selection(plugin, homoskedastic)`

  The errors must be homoskedastic, but no specific distribution is assumed.

  The formula for λ^* is

  $$\lambda_{\text{homoskedastic}} = \frac{c\widehat{\sigma}}{\sqrt{N}}\,\Phi^{-1}\left(1 - \frac{\gamma}{2p}\right)$$

  $\widehat{\sigma}$ is an estimator of the variance of the error term. This estimator is implemented in algorithm 1. In the linear homoskedastic case, there is no need to estimate the penalty loadings κ_j; they are implied by $\widehat{\sigma}$.

- `selection(plugin, heteroskedastic)`

  The errors may be heteroskedastic and no specific distribution is assumed.

  The formula for λ is

  $$\lambda_{\text{heteroskedastic}} = \frac{c}{\sqrt{N}}\,\Phi^{-1}\left(1 - \frac{\gamma}{2p}\right)$$

  In the linear-heteroskedastic case, penalty loadings are estimated by

  $$\kappa_j = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{ij}\widehat{\epsilon}_i)^2}$$

  Algorithm 2 is used to estimate the ϵ_i.

One plugin estimator is implemented for `lasso logit`:

$$\lambda_{\text{logit}} = \frac{c}{2\sqrt{N}}\,\Phi^{-1}\left[1 - \frac{1.1}{2\max\{N, p\ln N\}}\right]$$

This value is from the notes to table 2 in Belloni, Chernozhukov, and Wei (2016), divided by N as noted above. Belloni, Chernozhukov, and Wei (2016) use the structure of the binary model to bound the κ_j, so they are not estimated. This bound is why c is divided by 2.

One plugin estimator is implemented for `lasso poisson` and `lasso probit`:

$$\lambda = \frac{c}{\sqrt{N}}\,\Phi^{-1}\left(1 - \frac{\gamma}{2p}\right)$$

κ_j are estimated in algorithm 3.

All three algorithms used the normalized covariates that each $\mathbf{x}_j$ has mean 0 and variance 1.

Algorithm 1: Estimate $\widehat{\sigma}$

This iterative algorithm estimates σ; it is adopted from Belloni and Chernozhukov (2011, 20–21). The algorithm depends on a starting value for $\widehat{\sigma}$ denoted by $\widehat{\sigma}_0$, a convergence tolerance $v = 1\text{e}{-8}$, and a maximum number of iterations $M = 15$.

We set $\widehat{\sigma}_0$ to be the square root of the mean of the squared residuals from a regression of y on the five covariates in $\mathbf{x}$ that have the five highest univariate correlations with y.

Set the iteration counter $k = 1$ and the absolute value of the difference between the current and the previous estimate of σ to be a missing value.

1. Let $\widehat{\lambda}_k = (c\widehat{\sigma}_{k-1}/\sqrt{N})\,\Phi^{-1}(1-\gamma/2p)$.
2. Compute the lasso estimates $\widehat{\boldsymbol{\beta}}_k$ using $\widehat{\lambda}_k$.
3. Let $\widehat{\sigma}_k = \sqrt{(1/N)\,\sum_{i=1}^{N}(y_i - \mathbf{x}_i\widehat{\boldsymbol{\beta}}_k)^2}$.
4. If $|\widehat{\sigma}_k - \widehat{\sigma}_{k-1}| < v$ or $k > M$, set $\widehat{\sigma} = \widehat{\sigma}_k$ and stop; otherwise, set $k = k+1$ and go to step 1.

Algorithm 2: Estimate linear-heteroskedastic penalty loadings

This iterative algorithm estimates the penalty loadings κ_j for the linear-heteroskedastic model; it is adopted from Belloni, Chernozhukov, and Hansen (2014b, 640). The algorithm depends on a convergence tolerance $v = 1\text{e}{-8}$ and a maximum number of iterations $M = 15$.

1. Get initial values:
   a. Let $\widehat{\epsilon}_0$ be the residuals from the regression of y on the five covariates in $\mathbf{x}$ that have the highest univariate correlations with y.
   b. Let $\widehat{\kappa}_{0,j} = \sqrt{1/N\sum_{i=1}^{N}(x_{i,j}\widehat{\epsilon}_k)^2}$ be the initial penalty loading for each covariate j.
   c. Let $\widehat{\lambda} = c/\sqrt{N}\,\Phi^{-1}(1-\gamma/2p)$.
   d. Set the iteration counter to $k = 1$.
2. Compute the lasso estimates $\widehat{\boldsymbol{\beta}}_k$ using $\widehat{\lambda}$ and the penalty loadings $\widehat{\kappa}_{k-1,j}$. Let $\widehat{s}$ be the number of nonzero coefficients in this lasso.

3. Let $\widehat{\epsilon}_k$ be the residuals from the postlasso regression of y on the $\widehat{s}$ covariates that have nonzero lasso coefficients.
4. For each of the j covariates in the original model, compute the penalty loading

$$\widehat{\kappa}_{k,j} = \sqrt{\frac{1}{N-\widehat{s}}\sum_{i=1}^{N}(x_{ij}\widehat{\epsilon}_k)^2}$$

5. If $\max_{1\leq j\leq p}|\widehat{\kappa}_{k,j} - \widehat{\kappa}_{k-1,j}| < v$ or $k > M$, set $\widehat{\kappa}_j = \widehat{\kappa}_{k,j}$ for each j and stop; otherwise, set $k = k+1$ and go to step 2.

Algorithm 3: Estimate penalty loadings for Poisson and probit

This is the algorithm used for Poisson and probit models.

In the Poisson case, references to the unpenalized quasi–maximum likelihood (QML) estimator are to the unpenalized Poisson QML estimator. In the probit case, references to the unpenalized QML estimator are to the unpenalized probit QML estimator.

In the Poisson case, $\mathbf{h}_j(y_i, \tilde{\mathbf{x}}_i\widetilde{\boldsymbol{\beta}})$ is the contribution of observation i to the unpenalized Poisson-score equation using covariates $\tilde{\mathbf{x}}_i$ and coefficients $\widetilde{\boldsymbol{\beta}}$. In the probit case, $\mathbf{h}_j(y_i, \tilde{\mathbf{x}}_i\widetilde{\boldsymbol{\beta}})$ is the contribution of observation i to the unpenalized probit-score equation using covariates $\tilde{\mathbf{x}}_i$ and coefficients $\widetilde{\boldsymbol{\beta}}$.

On exit, λ contains the penalty value, and the penalty loadings are in $(\tilde{\kappa}_1, \ \ldots, \ \tilde{\kappa}_p)$.

1. Set $\lambda = c/\sqrt{N}\Phi^{-1}\left[1-\gamma/(2p)\right]$.
2. Find the five covariates with highest correlations with y. Denote the vector of them by $\tilde{\mathbf{x}}_0$, and let $\tilde{\mathbf{x}}_{0i}$ be the ith observation on this vector of variables.
3. Estimate the coefficients $\widetilde{\boldsymbol{\beta}}_0$ on $\tilde{\mathbf{x}}_0$ by unpenalized QML.
4. For each $j \in \{1, \ldots, p\}$, set

$$\tilde{\kappa}_{0,j} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\mathbf{h}_j(y_i, \tilde{\mathbf{x}}_{0i}\widetilde{\boldsymbol{\beta}}_0)^2}$$

5. Set $k = 1$ and do the following loop. (It will be executed at most 15 times.)
   a. Using λ and loadings $\{\tilde{\kappa}_{k-1,1}, \ \ldots, \ \tilde{\kappa}_{k-1,p}\}$, solve the lasso to get estimates $\widetilde{\widetilde{\boldsymbol{\beta}}}_k$.
   b. Let $\tilde{\mathbf{x}}_k$ be the covariates with nonzero coefficients in $\widetilde{\widetilde{\boldsymbol{\beta}}}_k$.
   c. Estimate the coefficients $\widetilde{\boldsymbol{\beta}}_k$ on $\tilde{\mathbf{x}}_k$ by unpenalized QML.
   d. For each $j \in \{1, \ldots, p\}$, set

$$\tilde{\kappa}_{k,j} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\mathbf{h}_j(y_i, \tilde{\mathbf{x}}_{ki}\widetilde{\boldsymbol{\beta}}_k)^2}$$

   e. Set $k = k+1$.
   f. If $k > 15$ or the variables in $\tilde{\mathbf{x}}_k$ are the same as those in $\tilde{\mathbf{x}}_{k-1}$, set each $\tilde{\kappa}_j = \tilde{\kappa}_{k,j}$ and exit; otherwise, go to step 5a.

BIC

lasso and elasticnet compute the BIC function for each vector of coefficients corresponding to each λ. The BIC function is defined as

$$\text{BIC} = -2\,\ln L(y, \beta_0 + \mathbf{x}\boldsymbol{\beta}') + k\,\ln N$$

where $\ln L(y, \beta_0 + \mathbf{x}\boldsymbol{\beta}')$ is the log-likelihood function, k is the number of nonzero coefficients, and N is the number of observations.

When the model is linear,

$$\ln L(y, \beta_0 + \mathbf{x}\boldsymbol{\beta}') = -\frac{1}{2}\left[\ln 2\pi + \ln\left\{\sum_{i=1}^{N} w_i^*(y_i - \beta_0 - \mathbf{x}_i\boldsymbol{\beta}')^2\right\} + 1\right]$$

When the model is logit,

$$\ln L(y, \beta_0 + \mathbf{x}\boldsymbol{\beta}') = \sum_{i=1}^{N} w_i^*\left[y_i\left(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}'\right) - \ln\left\{1 + \exp(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')\right\}\right]$$

When the model is probit,

$$\ln L(y, \beta_0 + \mathbf{x}\boldsymbol{\beta}') = \sum_{i=1}^{N} w_i^*\left[y_i \ln\left\{\Phi(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')\right\} + (1 - y_i)\ln\left\{1 - \Phi(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')\right\}\right]$$

When the model is poisson,

$$\ln L(y, \beta_0 + \mathbf{x}\boldsymbol{\beta}') = \sum_{i=1}^{N} w_i^*\left\{-\exp(\beta_0 + \mathbf{x}_i\boldsymbol{\beta}') + (\beta_0 + \mathbf{x}_i\boldsymbol{\beta}')y_i - \ln(y_i!)\right\}$$

When the model is cox,

$$\ln L(y, \mathbf{x}\boldsymbol{\beta}') = -\sum_{j=1}^{N_f}\sum_{i\in D_j} w_i^*\left[\mathbf{x}_i\boldsymbol{\beta}' - \ln\left\{\sum_{\ell\in R_j} w_\ell^*\exp(\mathbf{x}_\ell\boldsymbol{\beta}')\right\}\right]$$

The weights w_i^* are normalized to sum to N. That is,

$$w_i^* = \frac{N w_i}{\sum_{i=1}^{N} w_i}$$

where w_i is the original observation-level weight.

When the selection(bic, postselection) option is specified, the postselection coefficients are used to compute the BIC. By default, penalized coefficients are used.

References

Belloni, A., D. Chen, V. Chernozhukov, and C. B. Hansen. 2012. Sparse models and methods for optimal instruments with an application to eminent domain. *Econometrica* 80: 2369–2429. https://doi.org/10.3982/ECTA9626.

Belloni, A., and V. Chernozhukov. 2011. "High dimensional sparse econometric models: An Introduction". In *Inverse Problems of High-Dimensional Estimation*, edited by P. Alguier, E. Gautier, and G. Stoltz, 121–156. Berlin: Springer. https://doi.org/10.1007/978-3-642-19989-9_3.

Belloni, A., V. Chernozhukov, and C. B. Hansen. 2014a. High-dimensional methods and inference on structural and treatment effects. *Journal of Economic Perspectives* 28: 29–50. https://doi.org/10.1257/jep.28.2.29.

———. 2014b. Inference on treatment effects after selection among high-dimensional controls. *Review of Economic Studies* 81: 608–650. https://doi.org/10.1093/restud/rdt044.

Belloni, A., V. Chernozhukov, and Y. Wei. 2016. Post-selection inference for generalized linear models with many controls. *Journal of Business and Economic Statistics* 34: 606–619. https://doi.org/10.1080/07350015.2016.1166116.

Bickel, P. J., Y. Ritov, and A. B. Tsybakov. 2009. Simultaneous analysis of Lasso and Dantzig selector. *Annals of Statistics* 37: 1705–1732. https://doi.org/10.1214/08-AOS620.

Breslow, N. E. 1974. Covariance analysis of censored survival data. *Biometrics* 30: 89–99. https://doi.org/10.2307/2529620.

Bühlmann, P., and S. van de Geer. 2011. *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Berlin: Springer.

Chernozhukov, V., D. Chetverikov, M. Demirer, E. Duflo, C. B. Hansen, W. K. Newey, and J. M. Robins. 2018. Double/debiased machine learning for treatment and structural parameters. *Econometrics Journal* 21: C1–C68. https://doi.org/10.1111/ectj.12097.

Chetverikov, D., Z. Liao, and V. Chernozhukov. 2019. On cross-validated lasso in high dimensions. arXiv:1605.02214 [math.ST], https://doi.org/10.48550/arXiv.1605.02214.

Daubechies, I., M. Defrise, and C. D. Mol. 2004. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics* 57: 1413–1457. https://doi.org/10.1002/cpa.20042.

Friedman, J. H., T. J. Hastie, H. Höfling, and R. J. Tibshirani. 2007. Pathwise coordinate optimization. *Annals of Applied Statistics* 1: 302–332. https://doi.org/10.1214/07-AOAS131.

Friedman, J. H., T. J. Hastie, and R. J. Tibshirani. 2010. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software* 33: art. 1. https://doi.org/10.18637/jss.v033.i01.

Fu, W. J. 1998. Penalized regressions: The bridge versus the lasso. *Journal of Computational and Graphical Statistics* 7: 397–416. https://doi.org/10.1080/10618600.1998.10474784.

Hastie, T. J., R. J. Tibshirani, and M. Wainwright. 2015. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Boca Raton, FL: CRC Press. https://doi.org/10.1201/b18401.

Leeb, H., and B. M. Pötscher. 2008. Sparse estimators and the oracle property, or the return of Hodges' estimator. *Journal of Econometrics* 142: 201–211. https://doi.org/10.1016/j.jeconom.2007.05.017.

Nelder, J. A., and R. W. M. Wedderburn. 1972. Generalized linear models. *Journal of the Royal Statistical Society,* A ser., 135: 370–384. https://doi.org/10.2307/2344614.

Quandt, R. E. 1984. "Computational problems and methods". In *Handbook of Econometrics*, edited by Z. Griliches and M. D. Intriligator, vol. 2: 699–764. Amsterdam: Elsevier. https://doi.org/10.1016/S1573-4412(83)01016-8.

van Houwelingen, H. C., T. Bruinsma, A. A. M. Hart, L. J. van't Veer, and L. F. A. Wessels. 2006. Cross-validated Cox regression on microarray gene expression data. *Statistics in Medicine* 25: 3201–3216. https://doi.org/10.1002/sim.2353.

Zhang, Y., R. Li, and C.-L. Tsai. 2010. Regularization parameter selections via generalized information criterion. *Journal of the American Statistical Association* 105: 312–323. https://doi.org/10.1198/jasa.2009.tm08013.

Zou, H. 2006. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association* 101: 1418–1429. https://doi.org/10.1198/016214506000000735.

Also see

[LASSO] **lasso postestimation** — Postestimation tools for lasso for prediction

[LASSO] **elasticnet** — Elastic net for prediction and model selection

[LASSO] **lasso examples** — Examples of lasso for prediction

[LASSO] **Lasso intro** — Introduction to lasso

[LASSO] **sqrtlasso** — Square-root lasso for prediction and model selection

[R] **logit** — Logistic regression, reporting coefficients

[R] **poisson** — Poisson regression

[R] **probit** — Probit regression

[R] **regress** — Linear regression

[ST] **stset** — Declare data to be survival-time data

[U] 20 Estimation and postestimation commands

lasso postestimation — Postestimation tools for lasso for prediction

Postestimation commands

The following postestimation commands are of special interest after `lasso`, `sqrtlasso`, and `elasticnet`:

Command	Description
`bicplot`	plot Bayesian information criterion function
`coefpath`	plot path of coefficients
`cvplot`	plot cross-validation function
`lassocoef`	display selected coefficients
`lassogof`	goodness of fit after lasso for prediction
`lassoinfo`	information about lasso estimation results
`lassoknots`	knot table of coefficient selection and measures of fit
`lassoselect`	select alternative λ^* (and α^* for `elasticnet`)
* `stcurve`	plot the survivor, failure, hazard, or cumulative hazard function

*`stcurve` is appropriate only after `lasso cox` or `elasticnet cox`.

The following standard postestimation commands are also available:

Command	Description
`estat summarize`	summary statistics for the estimation sample
`estimates`	cataloging estimation results
`etable`	table of estimation results
`predict`	linear predictions

predict

Description for predict

`predict` creates a new variable containing predictions such as linear predictions; probabilities when the model is logit or probit; number of events when the model is Poisson; or hazard ratios and baseline survivor, cumulative hazard, and hazard functions when the model is Cox.

Menu for predict

Statistics > Postestimation

Syntax for predict

`predict` [*type*] *newvar* [*if*] [*in*] [, *statistic options*]

statistic	Description
Main	
`xb`	linear predictions; the default for the `linear` model
`pr`	probability of a positive outcome; the default for the `logit` and `probit` models
`n`	number of events; the default for the `poisson` model
`ir`	incidence rate; optional for the `poisson` model
`hr`	predicted hazard ratio, also known as the relative hazard; the default for the `cox` model
`basesurv`	baseline survivor function
`basechazard`	baseline cumulative hazard function
`basehc`	baseline hazard contributions

`pr` is allowed only when the model is `logit` or `probit`.

`n` and `ir` are allowed only when the model is `poisson`.

`hr`, `basesurv`, `basechazard`, and `basehc` are allowed only when the model is `cox`.

options	Description
Main	
`penalized`	use penalized coefficients; the default
`postselection`	use postselection (unpenalized) coefficients
`nooffset`	ignore the offset or exposure variable (if any)

Unstarred statistics are available both in and out of sample; type `predict ... if e(sample) ...` if wanted only for the estimation sample. Starred statistics are calculated only for the estimation sample, even when `e(sample)` is not specified. `nooffset` is allowed only with unstarred statistics.

Options for predict

Main

`xb`, the default for the `linear` model, calculates linear predictions.

`pr`, the default for and only allowed with the `logit` and `probit` models, calculates the probability of a positive event.

`n`, the default for and only allowed with the `poisson` model, calculates the number of events, which is $\exp(\mathbf{x}_j\boldsymbol{\beta})$ if neither `offset()` nor `exposure()` was specified when the model was fit; $\exp(\mathbf{x}_j\boldsymbol{\beta} + \text{offset}_j)$ if `offset()` was specified; or $\exp(\mathbf{x}_j\boldsymbol{\beta}) \times \text{exposure}_j$ if `exposure()` was specified.

`ir` applies to the `poisson` model only. It calculates the incidence rate $\exp(\mathbf{x}\boldsymbol{\beta}')$, which is the predicted number of events when exposure is 1. Specifying `ir` is equivalent to specifying `n` when neither `offset()` nor `exposure()` was specified when the model was fit.

`hr`, the default for the `cox` model, calculates the relative hazard (hazard ratio), that is, the exponentiated linear prediction $\exp(\mathbf{x}\boldsymbol{\beta}')$.

`basesurv` applies to the `cox` model only. It calculates the baseline survivor function. In the null model, this is equivalent to the Kaplan–Meier product-limit estimate.

`basechazard` applies to the `cox` model only. It calculates the cumulative baseline hazard.

`basehc` applies to the `cox` model only. It calculates the baseline hazard contributions. These are used to construct the product-limit type estimator for the baseline survivor function generated by `basesurv`.

`penalized` specifies that penalized coefficients be used to calculate predictions. This is the default. Penalized coefficients are those estimated by lasso in the calculation of the lasso penalty. See *Methods and formulas* in [LASSO] **lasso**.

`postselection` specifies that postselection coefficients be used to calculate predictions. Postselection coefficients are calculated by taking the variables selected by lasso and refitting the model with the appropriate ordinary estimator: linear regression for `linear` models, logistic regression for `logit` models, probit regression for `probit` models, Poisson regression for `poisson` models, and Cox regression for `cox` models.

`nooffset` is relevant only if you specified `offset()` or `exposure()` when you fit the model. It modifies the calculations made by `predict` so that they ignore the offset or exposure variable; the linear prediction is treated as $\mathbf{x}\boldsymbol{\beta}'$ rather than $\mathbf{x}\boldsymbol{\beta}' + \text{offset}$ or $\mathbf{x}\boldsymbol{\beta}' + \ln(\text{exposure})$. For the `poisson` model, specifying `predict ..., nooffset` is equivalent to specifying `predict ..., ir`. This option is not allowed when `basesurv`, `basechazard`, or `basehc` is specified.

stcurve

Description for stcurve

`stcurve` plots the survivor, failure, hazard, or cumulative hazard function after `lasso cox` or `elasticnet cox`.

Menu for stcurve

Statistics > Survival analysis > Regression models > Plot survivor or related function

Syntax for stcurve

`stcurve` [, `penalized` `postselection` *stcurve_options*]

Options for stcurve

`penalized`, the default, specifies that penalized coefficients be used to calculate predictions. Penalized coefficients are those estimated by lasso in the calculation of the lasso penalty. See *Methods and formulas* in [LASSO] **lasso**.

`postselection` specifies that postselection coefficients be used to calculate predictions. Postselection coefficients are calculated by taking the variables selected by lasso and refitting the model with `stcox`.

stcurve_options are options available for `stcurve`; see *Options* in [ST] **stcurve**.

Remarks and examples

By default, `predict` after `lasso` uses the penalized coefficient estimates to predict the outcome. Specifying the `postselection` option causes `predict` to use the postselection coefficients to calculate predictions. Postselection coefficients are calculated by taking the variables selected by lasso and refitting the model with the unpenalized estimator.

`stcurve` after `lasso cox` or `elasticnet cox` also uses the penalized coefficients by default. Specifying the `postselection` option causes `stcurve` to use the postselection coefficients.

Belloni and Chernozhukov (2013) and Belloni et al. (2012) provide results under which predictions using postselection coefficients perform at least as well as predictions using penalized coefficients. Their results are only for linear models. Their conditions essentially limit the cases to ones in which the covariates selected by the lasso are close to the set of covariates that best approximates the outcome. Said plainly, this means that under the conditions for which lasso provides valid predictions, the postselection coefficients should do slightly better than the penalized coefficients in most cases; in other cases, they should be about the same.

Rather than relying on theorems, standard practice in prediction applications uses split-sample techniques to find which of several models produces the best predictions. One standard practice in prediction applications is to randomly split the sample into training and testing samples. When you use the training data, the coefficients for several competing predictors are computed. When you use the testing data, an out-of-sample prediction error is computed for each of the predictors whose coefficients were estimated on the training data. The predictor with the smallest out-of-sample prediction error is preferred. This practice is illustrated in [LASSO] **lassogof**.

Methods and formulas

Below, we discuss the methods and formulas for the predictions of baseline survivor function, baseline cumulative hazard function, and baseline hazard contributions after `lasso cox` or `elasticnet cox`.

Define $z_i = \mathbf{x}_i\widehat{\boldsymbol\beta}' + \text{offset}_i$, where $\widehat{\boldsymbol\beta}$ is either the penalized or the postselection coefficients. The estimated baseline hazard contribution is obtained at each failure time as $h_j = 1 - \hat\alpha_j$, where $\hat\alpha_j$ is the solution to

$$\sum_{k\in D_j} \frac{\exp(z_k)}{1-\hat\alpha_j^{\exp(z_k)}} = \sum_{\ell\in R_j} \exp(z_\ell)$$

(Kalbfleisch and Prentice 2002, eq. 4.34, 115), where j indexes the ordered failure times t_j ($j = 1,\dots,D$); D_j is the set of d_j observations that fail at t_j; d_j is the number of failures at t_j; and R_j is the set of observations k that are at risk at time t_j (that is, all k such that $t_{0k} < t_j \le t_k$, and t_{0k} is the entry time for the kth observation).

The estimated baseline survivor function is

$$\hat S_0(t) = \prod_{j:t_j\le t} \hat\alpha_j$$

The estimated baseline cumulative hazard function, if requested, is related to the baseline survivor function calculation; yet the values of $\hat\alpha_j$ are set at their starting values and are not iterated. Equivalently,

$$\widehat H_0(t) = \sum_{j:t_j\le t} \frac{d_j}{\sum_{\ell\in R_j}\exp(z_\ell)}$$

For an application of this formula in the context of `lasso cox`, see Ternès, Rotolo, and Michiels (2017).

References

Belloni, A., D. Chen, V. Chernozhukov, and C. B. Hansen. 2012. Sparse models and methods for optimal instruments with an application to eminent domain. *Econometrica* 80: 2369–2429. https://doi.org/10.3982/ECTA9626.

Belloni, A., and V. Chernozhukov. 2013. Least squares after model selection in high-dimensional sparse models. *Bernoulli* 19: 521–547. https://doi.org/10.3150/11-BEJ410.

Kalbfleisch, J. D., and R. L. Prentice. 2002. *The Statistical Analysis of Failure Time Data*. 2nd ed. New York: Wiley.

Ternès, N., F. Rotolo, and S. Michiels. 2017. Robust estimation of the expected survival probabilities from high-dimensional Cox models with biomarker-by-treatment interactions in randomized clinical trials. *BMC Medical Research Methodology* 17(art. 83). https://doi.org/10.1186/s12874-017-0354-0.

Also see

[LASSO] **lasso examples** — Examples of lasso for prediction

[LASSO] **elasticnet** — Elastic net for prediction and model selection

[LASSO] **lasso** — Lasso for prediction and model selection

[LASSO] **sqrtlasso** — Square-root lasso for prediction and model selection

[U] 20 Estimation and postestimation commands

lassocoef — Display coefficients after lasso estimation results

Description Quick start Menu Syntax Options
Remarks and examples Stored results Also see

Description

lassocoef displays a table showing the selected variables after one or more lasso estimation results. It can also display the values of the coefficient estimates. When used with stored results from two or more lassos, it can be used to view the overlap between sets of selected variables.

After ds, po, and xpo commands, and after telasso, lassocoef can be used to view coefficients for a single lasso or for multiple lassos displayed side by side.

Quick start

Display the selected variables after lasso, sqrtlasso, or elasticnet

```
lassocoef
```

Display the values of the postselection coefficients after lasso, sqrtlasso, or elasticnet

```
lassocoef, display(coef, postselection)
```

Display the penalized coefficients of the standardized variables after lasso, sqrtlasso, or elasticnet sorted by their absolute values in descending order

```
lassocoef, display(coef, standardized) sort(coef, standardized)
```

Compare which variables were selected from three different runs of lasso, where the estimation results are stored under the names mylasso1, mylasso2, and mylasso3

```
lassocoef mylasso1 mylasso2 mylasso3
```

Same as above, but display the penalized coefficients of the unstandardized variables sorted by the values of the penalized coefficients of the standardized variables

```
lassocoef mylasso1 mylasso2 mylasso3, display(coef, penalized) ///
   sort(coef, standardized)
```

After fitting a lasso logit model, display the exponentiated postselection coefficients, which are odds ratios, and specify their display format

```
lassocoef, display(coef, postselection eform format(%6.2f))
```

After any of the ds or po commands, display the selected variables in the lasso for the dependent variable y

```
lassocoef (., for(y))
```

Same as above, but display the penalized coefficients of the standardized variables in the lasso for y sorted by their absolute values

```
lassocoef (., for(y)), display(coef, standardized) ///
   sort(coef, standardized)
```

Same as above, but compare the lasso for y from the results stored in mydsregress with the lasso for y from the results stored in myporegress

```
lassocoef (mydsregress, for(y)) (myporegress, for(y)), ///
   display(coef, standardized) sort(coef, standardized)
```

After xpologit without resample, compare the variables selected by the lassos for x in each of the 10 cross-fit folds

```
lassocoef (myxpo, for(x) xfold(1)) ///
      (myxpo, for(x) xfold(2)) ///
      ⋮
      (myxpo, for(x) xfold(10))
```

After xpologit with resample, compare the variables selected by the lassos for x in each of the 10 cross-fit folds in the first resample

```
lassocoef (myxpo, for(x) xfold(1) resample(1)) ///
      (myxpo, for(x) xfold(2) resample(1)) ///
      ⋮
      (myxpo, for(x) xfold(10) resample(1))
```

After telasso, display the selected variables in the lasso for the outcome variable y at treatment levels 1 and 0

```
lassocoef (., for(y) tlevel(1)) (., for(y) tlevel(0))
```

Menu

Statistics > Postestimation

Syntax

For current estimation results

After `lasso`, `sqrtlasso`, *or* `elasticnet`

`lassocoef` [, *options*]

After `ds` *or* `po`

`lassocoef (., for(`*varspec*`))` [, *options*]

After `xpo` *without* `resample`

`lassocoef (., for(`*varspec*`) xfold(#))` [, *options*]

After `xpo` *with* `resample`

`lassocoef (., for(`*varspec*`) xfold(#) resample(#))` [, *options*]

After `telasso` *for the outcome variable*

`lassocoef (., for(`*varspec*`) tlevel(#))` [, *options*]

After `telasso` *for the treatment variable*

`lassocoef (., for(`*varspec*`))` [, *options*]

After `telasso` *for the outcome variable with cross-fitting but without* `resample`

`lassocoef (., for(`*varspec*`) tlevel(#) xfold(#))` [, *options*]

After `telasso` *for the treatment variable with cross-fitting but without* `resample`

`lassocoef (., for(`*varspec*`) xfold(#))` [, *options*]

After `telasso` *for the outcome variable with cross-fitting and* `resample`

`lassocoef (., for(`*varspec*`) tlevel(#) xfold(#) resample(#))` [, *options*]

After `telasso` *for the treatment variable with cross-fitting and* `resample`

`lassocoef (., for(`*varspec*`) xfold(#) resample(#))` [, *options*]

For multiple stored estimation results

`lassocoef` [*estspec1* [*estspec2* ...]] [, *options*] *estspec* for `lasso`, `sqrtlasso`, and `elasticnet` is

name

estspec for `ds` and `po` models is

`(`*name*`, for(`*varspec*`))`

estspec for `xpo` without `resample` is

`(`*name*`, for(`*varspec*`) xfold(#))`

estspec for `xpo` with `resample` is

`(`*name*`, for(`*varspec*`) xfold(#) resample(#))`

estspec for the treatment model in `telasso` is

`(`*name*`, for(`*varspec*`))`

estspec for the outcome model at the treatment level # in `telasso` is

`(`*name*`, for(`*varspec*`) tlevel(#))`

estspec for the treatment model in `telasso` with cross-fitting but without `resample` is

`(`*name*`, for(`*varspec*`) xfold(#))`

estspec for the outcome model at the treatment level # in `telasso` with cross-fitting but without `resample` is

`(`*name*`, for(`*varspec*`) tlevel(#) xfold(#))`

estspec for the treatment model in `telasso` with `resample` is

`(`*name*`, for(`*varspec*`) xfold(#) resample(#))`

estspec for the outcome model at the treatment level # in `telasso` with `resample` is

`(`*name*`, for(`*varspec*`) tlevel(#) xfold(#) resample(#))`

name is the name of a stored estimation result. Either nothing or a period (`.`) can be used to specify the current estimation result. `_all` or `*` can be used to specify all stored estimation results when all stored results are `lasso`, `sqrtlasso`, or `elasticnet`.

varspec is *varname*, except after `poivregress` and `xpoivregress`, when it is either *varname* or `pred(`*varname*`)`.

options	Description
Options	
display(x)	indicate selected variables with an x; the default
display(u)	same as display(x), except variables unavailable to be selected indicated with a u
display(coef [, *coef_di_opts*])	display coefficient values
sort(none)	order of variables as originally specified; the default
sort(names)	order by the names of the variables
sort(coef [, *coef_sort_opts*])	order by the absolute values of the coefficients in descending order
nofvlabel	display factor-variable level values rather than value labels
nolegend	report or suppress table legend
nolstretch	do not stretch the width of the table to accommodate long variable names

collect is allowed; see **[U] 11.1.10 Prefix commands**.

nofvlabel, nolegend, and nolstretch do not appear in the dialog box.

coef_di_opts	Description
standardized	display penalized coefficients of standardized variables; the default
penalized	display penalized coefficients of unstandardized variables
postselection	display postselection coefficients of unstandardized variables
eform	display $\exp(b)$ rather than the coefficient b
format(*%fmt*)	use numerical format *%fmt* for the coefficient values

coef_sort_opts	Description
standardized	sort by penalized coefficients of standardized variables
penalized	sort by penalized coefficients of unstandardized variables
postselection	sort by postselection coefficients of unstandardized variables

Options

Options

display(*displayspec*) specifies what to display in the table. The default is display(x).

Blank cells in the table indicate that the corresponding variable was not selected by the lasso or was not specified in the model.

For some variables without fitted values, a code that indicates the reason for omission is reported in the table.

Empty levels of factors and interactions are coded with the letter e.

Base levels of factors and interactions are coded with the letter b. Base levels can be set on *alwaysvars* (variables always included in the lasso) but not on *othervars* (the set of variables from which lasso selects).

Variables omitted because of collinearity are coded with the letter `o`. Lasso does not label as omitted any *othervars* because of collinearity. Collinear variables are simply not selected. Variables in *alwaysvars* can be omitted because of collinearity. See *Remarks and examples* in [LASSO] **Collinear covariates**.

`display(x)` displays an `x` in the table when the variable has been selected by the lasso; that is, it has a nonzero coefficient.

`display(u)` is the same as `display(x)`, except that when a variable was not specified in the model, `u` (for unavailable) is displayed instead of a blank cell.

`display(coef` [`, standardized penalized postselection eform format(`*%fmt*`)`]`)` specifies that coefficient values be displayed in the table.

`standardized` specifies that the penalized coefficients of the standardized variables be displayed. This is the default when `display(coef)` is specified without options. Penalized coefficients of the standardized variables are the coefficient values used in the estimation of the lasso penalty. See *Methods and formulas* in [LASSO] **lasso**.

`penalized` specifies that the penalized coefficients of the unstandardized variables be displayed. Penalized coefficients of the unstandardized variables are the penalized coefficients of the standardized variables with the standardization removed.

`postselection` specifies that the postselection coefficients of the unstandardized variables be displayed. Postselection coefficients of the unstandardized variables are obtained by fitting an ordinary model (`regress` for `lasso linear`, `logit` for `lasso logit`, `probit` for `lasso probit`, and `poisson` for `lasso poisson`) using the selected variables. See *Methods and formulas* in [LASSO] **lasso**.

`eform` displays coefficients in exponentiated form. For each coefficient, $\exp(b)$ rather than b is displayed. This option can be used to display odds ratios or incidence-rate ratios after the appropriate estimation command.

`format(`*%fmt*`)` specifies the display format for the coefficients in the table. The default is `format(%9.0g)`.

`sort(`*sortspec*`)` specifies that the rows of the table be ordered by specification given by *sortspec*.

`sort(none)` specifies that the rows of the table be ordered by the order the variables were specified in the model specification. This is the default.

`sort(names)` orders rows alphabetically by the variable names of the covariates. In the case of factor variables, main effects and nonfactor variables are displayed first in alphabetical order. Then, all two-way interactions are displayed in alphabetical order, then, all three-way interactions, and so on.

`sort(coef` [`, standardized penalized postselection`]`)` orders rows in descending order by the absolute values of the coefficients. When results from two or more estimation results are displayed, results are sorted first by the ordering for the first estimation result with rows representing coefficients not in the first estimation result last. Within the rows representing coefficients not in the first estimation result, the rows are sorted by the ordering for the second estimation result with rows representing coefficients not in the first or second estimation results last. And so on.

`standardized` orders rows in descending order by the absolute values of the penalized coefficients of the standardized variables. This is the default when `sort(coef)` is specified without options.

`penalized` orders rows in descending order by the absolute values of the penalized coefficients of the unstandardized variables.

`postselection` orders rows in descending order by the absolute values of the postselection coefficients of the unstandardized variables.

`nofvlabel` displays factor-variable level numerical values rather than attached value labels. This option overrides the `fvlabel` setting. See [R] **set showbaselevels**.

`nolegend` specifies that the legend at the bottom of the table not be displayed. By default, it is shown.

`nolstretch` specifies that the width of the table not be automatically widened to accommodate long variable names. When `nolstretch` is specified, names are abbreviated to make the table width no more than 79 characters. The default, `lstretch`, is to automatically widen the table up to the width of the Results window. To change the default, use `set lstretch off`.

Required options for *estspec* after `telasso`, `ds`, `po`, and `xpo`:

`for(`*varspec*`)` specifies a particular lasso after `telasso` or after a `ds`, `po`, or `xpo` estimation command fit using the option `selection(cv)`, `selection(adaptive)`, or `selection(bic)`. For all commands except `poivregress` and `xpoivregress`, *varspec* is always *varname*.

For the `ds`, `po`, and `xpo` commands except `poivregress` and `xpoivregress`, *varspec* is either *depvar*, the dependent variable, or one of *varsofinterest* for which inference is done.

For `poivregress` and `xpoivregress`, *varspec* is either *varname* or `pred(`*varname*`)`. The lasso for *depvar* is specified with its *varname*. Each of the endogenous variables have two lassos, specified by *varname* and `pred(`*varname*`)`. The exogenous variables of interest each have only one lasso, and it is specified by `pred(`*varname*`)`.

For `telasso`, *varspec* is either the outcome variable or the treatment variable.

This option is required after `telasso` and after the `ds`, `po`, and `xpo` commands.

`xfold(#)` specifies a particular lasso after an `xpo` estimation command or after `telasso` when the option `xfolds(#)` was specified. For each variable to be fit with a lasso, K lassos are done, one for each cross-fit fold, where K is the number of folds. This option specifies which fold, where $\# = 1, 2, \dots, K$. `xfold(#)` is required after an `xpo` command and after `telasso` when the option `xfolds(#)` was specified.

`resample(#)` specifies a particular lasso after an `xpo` estimation command or after `telasso` fit using the option `resample(#)`. For each variable to be fit with a lasso, $R \times K$ lassos are done, where R is the number of resamples and K is the number of cross-fitting folds. This option specifies which resample, where $\# = 1, 2, \dots, R$. `resample(#)`, along with `xfold(#)`, is required after an `xpo` command and after `telasso` with resampling.

`tlevel(#)` specifies the lasso for the outcome variable at the specified treatment level after `telasso`. This option is required to refer to the outcome model after `telasso`.

Remarks and examples

`lassocoef` lists the variables selected by a lasso and optionally lists the values of their coefficients. It is useful for comparing the results of multiple lassos. It shows how much overlap there is among the sets of selected variables from the lassos.

By default, `lassocoef` indicates only whether a variable was selected, marking a selected variable with an `x`. The option `display(coef,` *coef_type*`)` can be used to display the values of the coefficients.

Lassos store three different types of coefficients (*coef_types*). We refer to them as standardized, penalized, and postselection.

Before a lasso is fit, the potential variables in the model are standardized so that they each have mean 0 and standard deviation 1. standardized refers to the coefficients of the standardized variables exactly as estimated by the minimization of the objective function.

penalized refers to the coefficients from the minimization of the objective function with the standardization unwound. standardized, strictly speaking, gives the penalized coefficients of the standardized variables. penalized gives the penalized coefficients of the unstandardized variables.

postselection coefficients are computed by taking the selected variables and, for a linear lasso, estimating an ordinary least-squares linear regression with them, and using those coefficients. For a logit lasso, a logistic regression gives the postselection coefficients; for a probit lasso, a probit regression gives them; and for a Poisson lasso, a Poisson regression gives them.

lassocoef also has a sort(coef, *coef_type*) option, which controls the order in which the variables are listed. The most useful ordering is sort(coef, standardized). It sorts the listing by the absolute values of the standardized coefficients with the largest displayed first. Variables with larger absolute values of their standardized coefficients take up a larger share of the lasso penalty, and so in this sense, they are "more important" for prediction than variables with smaller values.

▷ Example 1: lasso

We will show some uses of lassocoef after lasso.

Here is an example using lasso from [LASSO] **lasso examples**. We load the data and make the vl variable lists active.

```
. use https://www.stata-press.com/data/r19/fakesurvey_vl
(Fictitious survey data with vl)

. vl rebuild
Rebuilding vl macros ...
```

(*output omitted*)

We fit the lasso.

```
. lasso linear q104 $idemographics $ifactors $vlcontinuous, rseed(1234)
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .9090511   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 18.33331
  (output omitted)
Grid value 28:    lambda = .0737359   no. of nonzero coef. =  80
Folds: 1...5....10   CVF = 11.92887
... cross-validation complete ... minimum found
Lasso linear model                          No. of obs        =        914
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds   =         10

                                        No. of      Out-of-      CV mean
                                       nonzero       sample   prediction
       ID     Description      lambda    coef.    R-squared        error

        1     first lambda   .9090511        0      -0.0010     18.33331
       23    lambda before   .1174085       58       0.3543     11.82553
     * 24  selected lambda   .1069782       64       0.3547     11.81814
       25     lambda after   .0974746       66       0.3545      11.8222
       28      last lambda   .0737359       80       0.3487     11.92887

* lambda selected by cross-validation.
```

By default, after only one lasso, `lassocoef` lists the variables selected by the lasso.

```
. lassocoef

                   active

     0.gender        x
         0.q3        x
         0.q4        x
         0.q5        x
         2.q6        x
         0.q7        x
  (output omitted)
         q111        x
         q139        x
        _cons        x

Legend:
  b - base level
  e - empty cell
  o - omitted
  x - estimated
```

`lassocoef` is intended to be used to compare multiple lassos. So let's store the results of this lasso before we fit another. See [LASSO] **estimates store** for more on storing and saving lasso results.

```
. estimates store lassocv
```

We fit an adaptive lasso.

```
. lasso linear q104 $idemographics $ifactors $vlcontinuous,
> selection(adaptive) rseed(1234)
```

(output omitted)

```
Lasso linear model                          No. of obs         =        914
                                            No. of covariates  =        277
Selection: Adaptive                         No. of lasso steps =          2

Final adaptive step results

----------------------------------------------------------------------------
         |                                No. of      Out-of-      CV mean
         |                               nonzero       sample   prediction
      ID |     Description      lambda     coef.    R-squared        error
---------+------------------------------------------------------------------
      29 |    first lambda    52.54847         0      -0.0011      18.3349
      82 |   lambda before    .3794425        40       0.4077     10.84767
    * 83 | selected lambda    .3457338        41       0.4077     10.84764
      84 |    lambda after    .3150198        42       0.4074     10.85301
     128 |     last lambda    .0052548        62       0.3954     11.07398
----------------------------------------------------------------------------
* lambda selected by cross-validation in final adaptive step.

. estimates store lassoadaptive
```

Adaptive lasso selected 41 variables. Lasso selected 64. We can compare both the differences in selection and differences in the values of the coefficients. We use `lassocoef` with `display(coef, standardized)` to list the values of the standardized coefficients. We specify `sort(coef, standardized)` to sort them so that the largest ones in absolute value from the first lasso are shown first. The option `nofvlabel` means that numerical values for the factor-variable levels are displayed rather than value labels.

```
. lassocoef lassocv lassoadaptive, display(coef, standardized)
> sort(coef, standardized) nofvlabel nolegend

---------------------------------------
             |  lassocv  lassoadaptive
-------------+-------------------------
      0.q19  | -.8228234      -.9542076
      0.q88  |  .7464342       .8650972
     3.q156  | -.6770033       -.770628
      0.q48  | -.6055556      -.7086328
      0.q73  | -.5962807      -.7036719
      0.q85  | -.5855315       -.684066
        q31  |  .5843145       .7228376
     0.q101  |  .5565875       .6682665
             |
```

(output omitted)

```
      0.q75  | -.0056084
        q63  | -.0055279
      0.q55  | -.0054106
      0.q51  |  .0043129
      0.q77  |  .0019468
     0.q115  |  .0005097
       _cons |         0       3.55e-15
---------------------------------------
```

Most of the differences occur in the coefficients with the smallest absolute values.

Let’s fit another lasso. Note that we omitted the variable list `idemographics` from the potential variables this time.

```
. lasso linear q104 $ifactors $vlcontinuous, selection(cv) rseed(1234)
```

(output omitted)

```
Lasso linear model                          No. of obs        =        916
                                            No. of covariates =        269
Selection: Cross-validation                 No. of CV folds   =         10

----------------------------------------------------------------------------
         |                                No. of      Out-of-      CV mean
         |                               nonzero       sample   prediction
      ID |     Description      lambda     coef.    R-squared        error
---------+------------------------------------------------------------------
       1 |    first lambda    .9127278         0      -0.0020     18.33925
      24 |   lambda before    .1074109        57       0.3406     12.06842
    * 25 | selected lambda    .0978688        62       0.3407     12.06704
      26 |    lambda after    .0891744        70       0.3400     12.07962
      28 |     last lambda    .0740342        78       0.3361     12.15082
----------------------------------------------------------------------------
* lambda selected by cross-validation.

. estimates store lassocv2
```

The option `display(u)` puts a `u` next to the variables that were unavailable to be selected.

```
. lassocoef lassocv lassocv2, display(u)

-----------------------------------
             |  lassocv   lassocv2
-------------+---------------------
    0.gender |     x          u
        0.q3 |     x          u
        0.q4 |     x          u
        0.q5 |     x          u
             |
          q6 |
           2 |     x          x
           3 |                x
             |
```

(output omitted)

```
        q100 |
          No |                x
         q21 |                x
         q52 |                x
       _cons |     x          x
-----------------------------------
Legend:
  b - base level
  e - empty cell
  o - omitted
  x - estimated
  u - not selected for estimation
```

If `display(u)` was not specified, there would be empty space in place of the `u`’s. So this option is useful for distinguishing whether a variable was not selected or simply not included in the model specification.

◁

▷ Example 2: poivregress

We want to show you some differences that arise when you fit models containing endogenous variables using poivregress and xpoivregress.

We will not describe the data or the model here. See [LASSO] **Inference examples**.

We load the data,

```
. use https://www.stata-press.com/data/r19/mroz2, clear
```

set vl variable lists,

```
. vl create vars      = (kidslt6 kidsge6 age husage city exper)
note: $vars initialized with 6 variables.
. vl substitute vars2 = c.vars c.vars#c.vars
. vl create iv        = (huseduc motheduc fatheduc)
note: $iv initialized with 3 variables.
. vl substitute iv2   = c.iv c.iv#c.iv
```

and fit our model using poivregress.

```
. poivregress lwage (educ = $iv2), controls($vars2) selection(cv) rseed(12345)
Estimating lasso for lwage using cv
Estimating lasso for educ using cv
Estimating lasso for pred(educ) using cv
Partialing-out IV linear model     Number of obs                =        428
                                   Number of controls           =         27
                                   Number of instruments        =          9
                                   Number of selected controls  =         16
                                   Number of selected instruments =        4
                                   Wald chi2(1)                 =      11.10
                                   Prob > chi2                  =     0.0009

-----------------------------------------------------------------------------
             |               Robust
       lwage | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+---------------------------------------------------------------
        educ |   .0765154   .0229707     3.33   0.001     .0314936    .1215371
-----------------------------------------------------------------------------
Endogenous: educ
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
. estimates store poivregresscv
```

We use lassoinfo to see the lassos fit by poivregress.

```
. lassoinfo poivregresscv
    Estimate: poivregresscv
     Command: poivregress

----------------------------------------------------------------
             |                                             No. of
             |           Selection  Selection             selected
    Variable |     Model    method  criterion     lambda  variables
-------------+--------------------------------------------------
       lwage |    linear        cv    CV min.   .0353704          3
        educ |    linear        cv    CV min.   .0530428         10
  pred(educ) |    linear        cv    CV min.    .013186         12
----------------------------------------------------------------
```

We have two lassos for educ, the endogenous variable in the model. One is named educ and the other pred(educ). To compare the coefficient estimates for these two lassos, we type

```
. lassocoef (poivregresscv, for(educ)) (poivregresscv, for(pred(educ))),
> display(coef, standardized) sort(coef, standardized) nolegend

                       poivregresscv  poivregresscv
                                educ     pred(educ)

    c.huseduc#c.huseduc     1.047956

  c.motheduc#c.fatheduc     .5574474

    c.kidsge6#c.kidsge6    -.2293016       -.274782

    c.kidslt6#c.kidslt6     .1175937

      c.kidsge6#c.exper     .1087689       .2928483

  c.motheduc#c.motheduc     .0813009

   c.huseduc#c.fatheduc     .0411326

         c.city#c.exper     .0207999       .1020498

       c.husage#c.exper     .0077213

       c.kidsge6#c.city    -.0017114

                kidslt6                    .5342914

    c.kidslt6#c.kidsge6                   -.2364133

                kidsge6                   -.2129479
                 husage                   -.2091804

        c.husage#c.city                    .1396385

        c.exper#c.exper                    -.133589

      c.kidslt6#c.exper                   -.1322304

          c.city#c.city                    .1320515

       c.kidslt6#c.city                    .0237243

                  _cons            0       1.78e-15
```

◁

▷ Example 3: xporegress

The xpo commands fit many lassos. For each lasso fit by a po command, the corresponding xpo command fits xfolds(#) × resample(#) lassos. Cross-fitting randomly creates different divisions of the data for each resample. We expect that lasso will select different variables for different cross-fit folds and resamples. See [LASSO] **Inference examples** for a description of the data and model.

We load the data, set vl variable lists, fit our model using xporegress with the options xfolds(3) and resample(2), and then store the results with estimates store.

```
. use https://www.stata-press.com/data/r19/breathe, clear
(Nitrogen dioxide and attention)

. vl set
```

(output omitted)

```
. vl move (siblings_old siblings_young) vlcontinuous
note: 2 variables specified and 2 variables moved.
```

(output omitted)

```
. vl create mycontinuous   = vlcontinuous - (react no2_class)
note: $mycontinuous initialized with 10 variables.

. vl substitute mycontrols = i.vlcategorical mycontinuous

. xporegress react no2_class, controls($mycontrols) xfolds(3) resample(2)
> selection(cv) rseed(12345)

Resample 1 of 2 ...
Cross-fit fold 1 of 3 ...
Estimating lassos: 1.

Resample 1 of 2 ...
Cross-fit fold 2 of 3 ...
Estimating lassos: 1.

Resample 1 of 2 ...
Cross-fit fold 3 of 3 ...
Estimating lassos: 1.

Resample 2 of 2 ...
Cross-fit fold 1 of 3 ...
Estimating lassos: 1.

Resample 2 of 2 ...
Cross-fit fold 2 of 3 ...
Estimating lassos: 1.

Resample 2 of 2 ...
Cross-fit fold 3 of 3 ...
Estimating lassos: 1.

Cross-fit partialing-out            Number of obs              =      1,036
linear model                        Number of controls         =         32
                                    Number of selected controls =        27
                                    Number of folds in cross-fit =        3
                                    Number of resamples        =          2
                                    Wald chi2(1)               =      20.99
                                    Prob > chi2                =     0.0000

------------------------------------------------------------------------------
             |               Robust
       react | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
   no2_class |   2.332193   .5090902     4.58   0.000     1.334394    3.329991
------------------------------------------------------------------------------

Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.

. estimates store xpocv
```

For each cross-fit fold and each resample, `xporegress` fits lassos. It fit six lassos for the dependent variable, `react`, and six for the variable of interest, `no2_class`. To see how the variables selected differ for different folds and for different resamples, we type

```
. lassocoef (xpocv, for(react) resample(1) xfold(1))
>           (xpocv, for(react) resample(1) xfold(2))
>           (xpocv, for(react) resample(1) xfold(3))
>           (xpocv, for(react) resample(2) xfold(1))
>           (xpocv, for(react) resample(2) xfold(2))
>           (xpocv, for(react) resample(2) xfold(3))
>           , sort(coef, standardized)

-------------------------------------------------------------------------------
             |   xpocv     xpocv      xpocv      xpocv      xpocv      xpocv
             | react_1_1 react_2_1  react_3_1  react_1_2  react_2_2  react_3_2
-------------+-----------------------------------------------------------------
       grade |
        2nd  |     x         x          x          x          x          x
             |
         sex |
       Male  |     x         x          x          x          x          x
             |
       grade |
        4th  |     x         x          x          x          x          x
             |
         age |     x         x          x          x          x          x
             |
  feducation |
 University  |     x         x          x          x          x          x
             |
        age0 |     x         x          x                     x          x
             |
  meducation |
    Primary  |     x         x          x          x          x          x
             |
  breastfeed |
          2  |     x         x                                x
             |
    0.msmoke |     x         x                                x
             |
  feducation |
    Primary  |               x          x          x                     x
   <Primary  |               x                                x
             |
  sev_school |               x                                           x
             |
  meducation |
   <Primary  |               x          x                     x          x
             |
      (output omitted)
             |
0.overweight |                                                x
      precip |                                                           x
  green_home |                                                           x
-------------------------------------------------------------------------------
Legend:
  b - base level
  e - empty cell
  o - omitted
  x - estimated
```

Even though we had `lassocoef` display `x`'s, we specified the `sort(coef, standardized)` option so that the table is ordered by the most important variables from the lasso in the first column.

◁

Stored results

`lassocoef` stores the following in `r()`:

Macros	
`r(names)`	names of results used
Matrices	
`r(coef)`	matrix M: $n \times m$
	$M[i,\ j]$ $= i$th coefficient estimate for model j; $i = 1, \dots, n$; $j = 1, \dots, m$

Also see

[LASSO] **lasso postestimation** — Postestimation tools for lasso for prediction

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **lassoinfo** — Display information about lasso estimation results

[CAUSAL] **telasso postestimation** — Postestimation tools for telasso

lasso examples — Examples of lasso for prediction

Description

This entry contains more examples of lasso for prediction. It assumes you have already read [LASSO] **Lasso intro** and [LASSO] **lasso**.

Remarks and examples

Remarks are presented under the following headings:

Overview

In the examples of this entry, we use a dataset of a realistic size for lasso. It has 1,058 observations and 172 variables. Still, it is a little on the small side for lasso. Certainly, you can use lasso on datasets of this size, but lasso can also be used with datasets that have thousands or tens of thousands of variables.

The number of variables can even be greater than the number of observations. What is essential for lasso is that the set of potential variables contains a subset of variables that are in the true model (or something close to it) or are correlated with the variables in the true model.

As to how many variables there can be in the true model, we can say that the number cannot be greater than something proportional to $\sqrt{N}/\ln q$, where N is the number of observations, p is the number of potential variables, and $q = \max\{N, p\}$. We cannot, however, say what the constant of proportionality is. That this upper bound decreases with q can be viewed as the cost of performing covariate selection.

Using vl to manage variables

We will show how to use commands in the `vl` system to manage large numbers of variables. `vl` stands for “variable lists”. The idea behind it is that we might want to run a lasso with hundreds or thousands or tens of thousands of variables specified as potential variables. We do not want to have to type all these variable names.

Many times, we will have a mix of different types of variables. Some we want to treat as continuous. Some we want to treat as categorical and use factor-variable operators with them to create indicator variables for their categories. See [U] **11.4.3 Factor variables**.

The first goal of the `vl` system is to help us separate variables we want to treat as categorical from those we want to treat as continuous. The second goal of the system is to help us create named variable lists we can use as arguments to `lasso` or any other Stata command simply by referring to their names.

The purpose here is to illustrate the power of `vl`, not to explain in detail how it works or show all of its features. For that, see [D] **vl**.

We load the dataset we will use in these examples.

```
. use https://www.stata-press.com/data/r19/fakesurvey
(Fictitious survey data)
```

It is simulated data designed to mimic survey data. It has 1,058 observations and 172 variables.

```
. describe

Contains data from https://www.stata-press.com/data/r19/fakesurvey.dta
 Observations:         1,058                  Fictitious survey data
    Variables:           172                  14 Jun 2024 15:31
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
id              str8    %9s                   Respondent ID
gender          byte    %8.0g      gender     Gender
age             byte    %8.0g                 Age (y)
q1              byte    %10.0g                Question 1
q2              byte    %8.0g                 Question 2
q3              byte    %8.0g      yesno      Question 3
  (output omitted)
q160            byte    %8.0g      yesno      Question 160
q161            byte    %8.0g      yesno      Question 161
check8          byte    %8.0g                 Check 8
-------------------------------------------------------------------------------
Sorted by: id
```

The variables are a mix. Some we know are integer-valued scales that we want to treat as continuous variables in our models. There are a lot of 0/1 variables, and there are some with only a few categories that we will want to turn into indicator variables. There are some with more categories that we do not yet know whether to treat as categorical or continuous.

The first `vl` subcommand we run is `vl set`. Nonnegative integer-valued variables are candidates for use as factor variables. Because factor variables cannot be negative, any variable with negative values is classified as continuous. Any variable with noninteger values is also classified as continuous.

`vl set` has two options, `categorical(#)` and `uncertain(#)`, that allow us to separate out the nonnegative integer-valued variables into three named variable lists: `vlcategorical`, `vluncertain`, and `vlcontinuous`.

When the number of levels (distinct values), L, is

$$2 \leq L \leq \texttt{categorical(\#)}$$

the variable goes in vlcategorical. When

$$\texttt{categorical(\#)} < L \leq \texttt{uncertain(\#)}$$

the variable goes in vluncertain. When

$$L > \texttt{uncertain(\#)}$$

the variable goes in vlcontinuous.

The defaults are categorical(10) and uncertain(100). For our data, we do not like the defaults, so we change them. We specify categorical(4) and uncertain(19). We also specify the option dummy to create a variable list, vldummy, consisting solely of 0/1 variables. Let's run vl set with these options.

```
. vl set, categorical(4) uncertain(19) dummy

-------------------------------------------------------------------------------
                     |                   Macro's contents
                     |---------------------------------------------------------
Macro                |  # Vars   Description
---------------------+---------------------------------------------------------
System               |
  $vldummy           |      99   0/1 variables
  $vlcategorical     |      16   categorical variables
  $vlcontinuous      |      20   continuous variables
  $vluncertain       |      27   perhaps continuous, perhaps categorical variables
  $vlother           |       9   all missing or constant variables
-------------------------------------------------------------------------------
Notes

      1. Review contents of vlcategorical and vlcontinuous to ensure they are
         correct.  Type vl list vlcategorical and type vl list vlcontinuous.

      2. If there are any variables in vluncertain, you can reallocate them
         to vlcategorical, vlcontinuous, or vlother.  Type
         vl list vluncertain.

      3. Use vl move to move variables among classifications.  For example,
         type vl move (x50 x80) vlcontinuous to move variables x50 and x80 to
         the continuous classification.

      4. vlnames are global macros.  Type the vlname without the leading
         dollar sign ($) when using vl commands.  Example: vlcategorical not
         $vlcategorical.  Type the dollar sign with other Stata commands to
         get a varlist.
```

The vluncertain variable list contains all the variables we are not sure whether we want to treat as categorical or continuous. We use vl list to list the variables in vluncertain.

```
. vl list vluncertain

Variable | Macro           Values         Levels
---------+-----------------------------------------
     q12 | $vluncertain    integers >=0        5
     q18 | $vluncertain    integers >=0        7
     q23 | $vluncertain    integers >=0       10
     q27 | $vluncertain    integers >=0        8
     q28 | $vluncertain    integers >=0       15
     q35 | $vluncertain    integers >=0        7
     q39 | $vluncertain    integers >=0        5
     q54 | $vluncertain    integers >=0       10
     q63 | $vluncertain    integers >=0        7
     q66 | $vluncertain    integers >=0        5
     q80 | $vluncertain    integers >=0        5
     q81 | $vluncertain    integers >=0        5
     q92 | $vluncertain    integers >=0        5
     q93 | $vluncertain    integers >=0        7
     q99 | $vluncertain    integers >=0        5
    q103 | $vluncertain    integers >=0        7
    q107 | $vluncertain    integers >=0       18
    q111 | $vluncertain    integers >=0        7
    q112 | $vluncertain    integers >=0        7
    q119 | $vluncertain    integers >=0        8
    q120 | $vluncertain    integers >=0        7
    q124 | $vluncertain    integers >=0       14
    q127 | $vluncertain    integers >=0        5
    q132 | $vluncertain    integers >=0        7
    q135 | $vluncertain    integers >=0       10
    q141 | $vluncertain    integers >=0       12
    q157 | $vluncertain    integers >=0        7
---------------------------------------------------
```

We are going to have to go through these variables one by one and reclassify them. We know we have several seven-level Likert scales in these data. We tabulate one of them.

```
. tabulate q18
            Question 18 |      Freq.     Percent        Cum.
------------------------+-----------------------------------
   Very strongly disagree |        139       13.15       13.15
      Strongly disagree |        150       14.19       27.34
               Disagree |        146       13.81       41.15
Neither agree nor disagree |        146       13.81       54.97
                  Agree |        174       16.46       71.43
         Strongly agree |        146       13.81       85.24
     Very strongly agree |        156       14.76      100.00
------------------------+-----------------------------------
                  Total |      1,057      100.00
```

We look at all the variables with seven levels, and they are all Likert scales. We want to treat them as continuous in our models, so we move them out of `vluncertain` and into `vlcontinuous`.

```
. vl move (q18 q35 q63 q93 q103 q111 q112 q120 q132 q157) vlcontinuous
note: 10 variables specified and 10 variables moved.
──────────────────────────────
Macro           # Added/Removed
──────────────────────────────
$vldummy                     0
$vlcategorical               0
$vlcontinuous               10
$vluncertain               -10
$vlother                     0
──────────────────────────────
```

When variables are moved into a new `vl` system-defined variable list, they are automatically moved out of their current system-defined variable list.

In our examples, we have three variables we want to predict: `q104`, a continuous variable; `q106`, a 0/1 variable; and `q107`, a count variable. Because we are going to use the variables in `vlcategorical` and `vlcontinuous` as potential variables to select in our lassos, we do not want these dependent variables in these variable lists. We move them into `vlother`, which is intended as a place to put variables we do not want in our models.

```
. vl move (q104 q106 q107) vlother
note: 3 variables specified and 3 variables moved.
──────────────────────────────
Macro           # Added/Removed
──────────────────────────────
$vldummy                    -1
$vlcategorical               0
$vlcontinuous               -1
$vluncertain                -1
$vlother                     3
──────────────────────────────
```

Notice the parentheses around the variable names when we used `vl move`. The rule for `vl` is to use parentheses around variable names and to not use parentheses for variable-list names.

The system-defined variable lists are good for a general division of variables. But we need further subdivision for our models. We have four demographic variables, which are all categorical, but we want them included in all lasso models. So we create a user-defined variable list containing these variables.

```
. vl create demographics = (gender q3 q4 q5)
note: $demographics initialized with 4 variables.
```

We want to convert the variables in `vldummy` and `vlcategorical` into indicator variables. We create a new variable list, `factors`, containing the union of these lists. Because we want to handle the variables in `demographics` separately, we remove them from `factors`.

```
. vl create factors = vldummy + vlcategorical
note: $factors initialized with 114 variables.

. vl modify factors = factors - demographics
note: 4 variables removed from $factors.
```

The `vl substitute` command allows us to apply factor-variable operators to a variable list. We turn the variables in `demographics` and `factors` into factor variables.

```
. vl substitute idemographics = i.demographics
. vl substitute ifactors = i.factors
```

We are done using vl and we save our dataset. One nice feature of vl is that the variable lists are saved with the data.

```
. label data "Fictitious survey data with vl"
. save fakesurvey_vl
file fakesurvey_vl.dta saved
```

We are now ready to run some lassos.

Using splitsample

Well, almost ready. We want to evaluate our lasso predictions on a sample that we did not use to fit the lasso. So we decide to randomly split our data into two samples of equal sizes. We will fit models on one, and we will use the other to test their predictions.

Let's load the version of our dataset that contains our variable lists. We first increase maxvar because we are going to create thousands of interactions in a later example.

```
. clear all
. set maxvar 10000
. use https://www.stata-press.com/data/r19/fakesurvey_vl
(Fictitious survey data with vl)
```

Variable lists are not automatically restored. We have to run vl rebuild to make them active.

```
. vl rebuild
Rebuilding vl macros ...

-------------------------------------------------------------------------------
                  |                      Macro's contents
                  |------------------------------------------------------------
Macro             |  # Vars   Description
------------------+------------------------------------------------------------
System            |
  $vldummy        |      98   0/1 variables
  $vlcategorical  |      16   categorical variables
  $vlcontinuous   |      29   continuous variables
  $vluncertain    |      16   perhaps continuous, perhaps categorical variables
  $vlother        |      12   all missing or constant variables
User              |
  $demographics   |       4   variables
  $factors        |     110   variables
  $idemographics  |           factor-variable list
  $ifactors       |           factor-variable list
-------------------------------------------------------------------------------
```

We now use splitsample to generate a variable indicating the two subsamples.

```
. set seed 1234
. splitsample, generate(sample) nsplit(2)
. label define svalues 1 "Training" 2 "Testing"
. label values sample svalues
```

Lasso linear models

When fitting our lasso model, we can now specify variables succinctly using our `vl` variable lists. Variable lists are really global macros—we bet you already guessed this. Listing them under the header "Macro" in `vl` output was a real tip-off, right? Because they are global macros, when we use them as arguments in commands, we put a `$` in front of them.

We put parentheses around `idemographics`. This notation means that we want to force these variables into the model regardless of whether lasso wants to select them. See *Syntax* in [LASSO] **lasso**.

We also set the random-number seed using the `rseed()` option so that we can reproduce our results.

We fit `lasso` on the first subsample.

```
. lasso linear q104 ($idemographics) $ifactors $vlcontinuous
> if sample == 1, rseed(1234)

10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .8978025   no. of nonzero coef. =   4
Folds: 1...5....10   CVF = 16.93341
  (output omitted)
Grid value 23:    lambda = .1159557   no. of nonzero coef. =  74
Folds: 1...5....10   CVF = 12.17933
... cross-validation complete ... minimum found

Lasso linear model                          No. of obs        =        458
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds   =         10

--------------------------------------------------------------------------
         |                                No. of      Out-of-      CV mean
         |                               nonzero       sample   prediction
      ID |     Description      lambda     coef.    R-squared        error
---------+----------------------------------------------------------------
       1 |    first lambda    .8978025         4       0.0147     16.93341
      18 |   lambda before    .1846342        42       0.2953     12.10991
    * 19 | selected lambda    .1682318        49       0.2968     12.08516
      20 |    lambda after    .1532866        55       0.2964     12.09189
      23 |     last lambda    .1159557        74       0.2913     12.17933
--------------------------------------------------------------------------
* lambda selected by cross-validation.

. estimates store linearcv
```

After the command finished, we used `estimates store` to store the results in memory so that we can later compare these results with those from other lassos. Note, however, that `estimates store` only saves them in memory. To save the results to disk, use

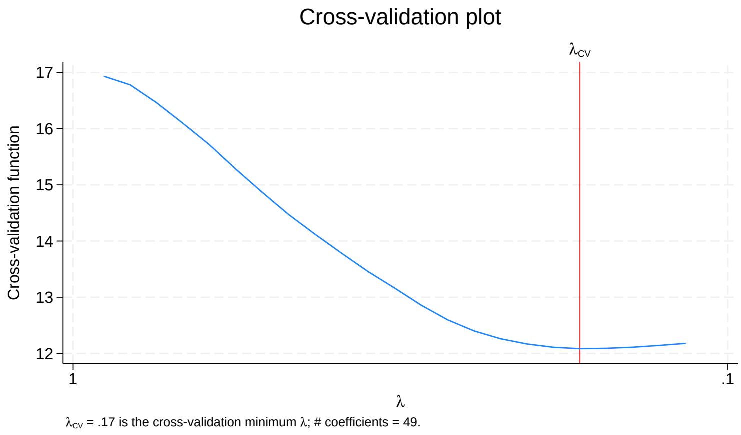
```
. estimates save filename
```

See [LASSO] **estimates store**.

The minimum of the cross-validation (CV) function was found to be at $\lambda = 0.1682318$. It selects λ^* as this λ, which corresponds to 49 variables in the model, out of 277 potential variables.

After fitting a lasso using CV to select λ, it is a good idea to plot the CV function and look at the shape of the curve around the minimum.

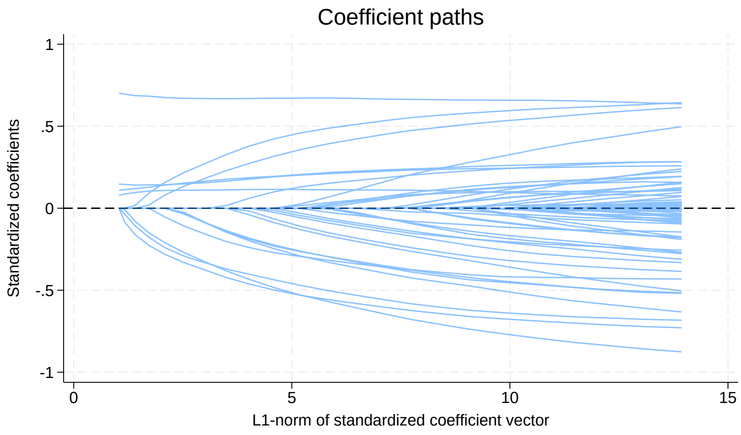
```
. cvplot
```



By default, the `lasso` command stops when it has identified a minimum. Computation time increases as λ's get smaller, so computing the CV function for smaller λ's is computationally expensive. We could specify the option `selection(cv, alllambdas)` to compute models for more small λ's. See [LASSO] **lasso** and [LASSO] **lasso fitting** for details and a description of less computationally intensive options to get more assurance that `lasso` has identified a minimum.

We can also get a plot of the size of the coefficients as they become nonzero and change as λ gets smaller. Typically, they get larger as λ gets smaller. But they can sometimes return to 0 after being nonzero.

```
. coefpath
```



We see four lines that do not start at 0. These are lines corresponding to the four variables in `idemographics` that we forced into the model.

Adaptive lasso

We are now going to run an adaptive lasso, which we do by specifying the option `selection(adaptive)`.

```
. lasso linear q104 ($idemographics) $ifactors $vlcontinuous
> if sample == 1, rseed(4321) selection(adaptive)
Lasso step 1 of 2:
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .8978025   no. of nonzero coef. =   4
Folds: 1...5....10   CVF =   17.012
  (output omitted)
Grid value 24:    lambda = .1056545   no. of nonzero coef. =  78
Folds: 1...5....10   CVF = 12.40012
... cross-validation complete ... minimum found
Lasso step 2 of 2:
Evaluating up to 100 lambdas in grid ...
Grid value 1:     lambda = 48.55244   no. of nonzero coef. =  4
  (output omitted)
Grid value 100:   lambda = .0048552   no. of nonzero coef. = 59
10-fold cross-validation with 100 lambdas ...
Fold  1 of 10:  10....20....30....40....50....60....70....80....90....100
  (output omitted)
Fold 10 of 10:  10....20....30....40....50....60....70....80....90....100
... cross-validation complete
Lasso linear model                        No. of obs         =        458
                                          No. of covariates  =        277
Selection: Adaptive                       No. of lasso steps =          2
Final adaptive step results

                                             No. of      Out-of-     CV mean
                                            nonzero       sample  prediction
       ID       Description       lambda      coef.    R-squared       error

       25      first lambda     48.55244          4       0.0101    17.01083
       77     lambda before     .3847698         46       0.3985    10.33691
     * 78   selected lambda     .3505879         46       0.3987    10.33306
       79      lambda after     .3194427         47       0.3985    10.33653
      124       last lambda     .0048552         59       0.3677    10.86697

* lambda selected by cross-validation in final adaptive step.
. estimates store linearadaptive
```

Adaptive lasso performs multiple lassos. In the first lasso, a λ^* is selected, and penalty weights are constructed from the coefficient estimates. Then these weights are used in a second lasso, where another λ^* is selected. We did not specify how many lassos should be performed, so we got the default of two. We could specify more, but typically the selected λ^* does not change after the second lasso, or it changes little. See the `selection(adaptive)` option in [LASSO] **lasso**.

We can see details of the two lassos by using lassoknots and specifying the option steps to see all steps of the adaptive lasso.

```
. lassoknots, steps

                                    No. of      CV mean
                                   nonzero        pred.    Variables (A)dded, (R)emoved,
Step       ID       lambda           coef.        error        or left (U)nchanged

   1
            1     .8978025               4       17.012    A 1.q3       1.q4
                                                             1.q5       1.gender
            2     .8180442               7     16.91096    A 0.q19      0.q85
                                                             3.q156
            3     .7453714               8     16.66328    A 0.q101
            4     .6791547               9     16.33224    A 0.q88
  (output omitted)
           23     .1159557              74     12.35715    A 3.q6       0.q40
                                                             0.q82      0.q98
                                                             0.q128     2.q134
                                                             0.q148     q157
           24     .1056545              78     12.40012    A 2.q6       0.q9
                                                             1.q34      4.q155

   2
           25     48.55244               4     17.01083    A 1.q3       1.q4
                                                             1.q5       1.gender
           26     44.23918               6     16.94087    A 0.q19      0.q85
  (output omitted)
           76     .4222844              45     10.33954    A 0.q44
           77     .3847698              46     10.33691    A q111
         * 78     .3505879              46     10.33306    U
           79     .3194427              47     10.33653    A 0.q97
           80     .2910643              48      10.3438    A 0.q138
  (output omitted)
          112     .0148272              59      10.7663    A q70
          124     .0048552              59     10.86697    U

* lambda selected by cross-validation in final adaptive step.
```

Notice how the scale of λ changes in the second lasso. That is because of the penalty weights generated by the first lasso.

The ordinary lasso selected 49 variables, and the adaptive lasso selected 46. It is natural to ask how much these two groups of variables overlap. When the goal is prediction, however, we are not supposed to care about this. Ordinary lasso might select one variable, and adaptive lasso might instead select another that is highly correlated to it. So it is wrong to place importance on any particular variable selected or not selected. It is the group of variables selected as a whole that matters.

Still, we cannot resist looking, and the lassocoef command was designed especially for this purpose. We specify lassocoef with the option sort(coef, standardized). This sorts the listing by the absolute values of the standardized coefficients with the largest displayed first. lassocoef can list different types of coefficients and display them in different orderings. See [LASSO] **lassocoef**.

```
. lassocoef linearcv linearadaptive, sort(coef, standardized)

─────────────────────────────────────────
             │  linearcv  linearadaptive
─────────────┼───────────────────────────
         q19 │
          No │         x             x
             │
         q85 │
          No │         x             x
             │
          q5 │
         Yes │         x             x
             │
      3.q156 │         x             x
             │
        q101 │
          No │         x             x
             │
  (output omitted)
             │
        q160 │
          No │         x             x
         age │         x             x
         q53 │         x             x
      2.q105 │         x
             │
        q102 │
          No │         x             x
             │
        q154 │
          No │         x             x
        q111 │         x             x
             │
        q142 │
          No │         x             x
       0.q55 │         x
       0.q97 │         x
             │
         q65 │
           4 │         x             x
             │
      1.q110 │         x             x
         q70 │         x
             │
         q44 │
          No │                       x
  (output omitted)
─────────────────────────────────────────
Legend:
  b - base level
  e - empty cell
  o - omitted
  x - estimated
```

We see that the adaptive lasso did not select four variables that the lasso did, and it selected one that the lasso did not. All the differences occurred among the variables with smaller standardized coefficients.

The most important question to ask is which performed better for out-of-sample prediction. `lassogof` is the command for that. We specify the `over()` option with the name of our sample indicator variable, `sample`. We specify the `postselection` option because for linear models, postselection coefficients are theoretically slightly better for prediction than the penalized coefficients (which `lassogof` uses by default).

```
. lassogof linearcv linearadaptive, over(sample) postselection
Postselection coefficients

Name                  sample         MSE    R-squared        Obs

linearcv
                    Training    8.652771       0.5065        503
                     Testing    14.58354       0.2658        493

linearadaptive
                    Training    8.637575       0.5057        504
                     Testing    14.70756       0.2595        494
```

The ordinary lasso did a little better in this case than the adaptive lasso.

Cross-validation folds

CV works by dividing the data randomly into K folds. One fold is chosen, and then a linear regression is fit on the other $K-1$ folds using the variables in the model for that λ. Then using these new coefficient estimates, a prediction is computed for the data of the chosen fold. The mean squared error (MSE) of the prediction is computed. This process is repeated for the other $K-1$ folds. The K MSEs are then averaged to give the value of the CV function.

Let's increase the number of folds from the default of 10 to 20 by specifying `selection(cv, folds(20))`.

```
. lasso linear q104 ($idemographics) $ifactors $vlcontinuous
> if sample == 1, selection(cv, folds(20)) rseed(9999)
20-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .8978025   no. of nonzero coef. =   4
Folds: 1...5....10....15....20   CVF = 17.08362
  (output omitted)
Grid value 23:    lambda = .1159557   no. of nonzero coef. =  74
Folds: 1...5....10....15....20   CVF = 12.12667
... cross-validation complete ... minimum found
Lasso linear model                          No. of obs        =        458
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds   =         20

                                              No. of      Out-of-      CV mean
                                             nonzero       sample   prediction
      ID     Description      lambda           coef.    R-squared        error

       1     first lambda    .8978025              4       0.0059     17.08362
      19    lambda before    .1682318             49       0.2999     12.03169
    * 20  selected lambda    .1532866             55       0.3002     12.02673
      21     lambda after     .139669             62       0.2988     12.05007
      23      last lambda    .1159557             74       0.2944     12.12667

* lambda selected by cross-validation.
. estimates store linearcv2
```

Which performs better for out-of-sample prediction?

```
. lassogof linearcv linearcv2, over(sample) postselection
Postselection coefficients
────────────────────────────────────────────────────────────────────
Name                 sample │            MSE     R-squared        Obs
────────────────────────────┼────────────────────────────────────────
linearcv                    │
                   Training │       8.652771        0.5065        503
                    Testing │       14.58354        0.2658        493
────────────────────────────┼────────────────────────────────────────
linearcv2                   │
                   Training │       8.545785        0.5126        502
                    Testing │        14.7507        0.2594        488
────────────────────────────────────────────────────────────────────
```

The first lasso with 10 folds did better than the lasso with 20 folds. This is generally true. More than 10 folds typically does not yield better predictions.

We should mention again that CV is a randomized procedure. Changing the random-number seed can result in a different λ^* being selected and so give different predictions.

BIC

We are now going to select λ^* by minimizing the BIC function, which we do by specifying the option `selection(bic)`.

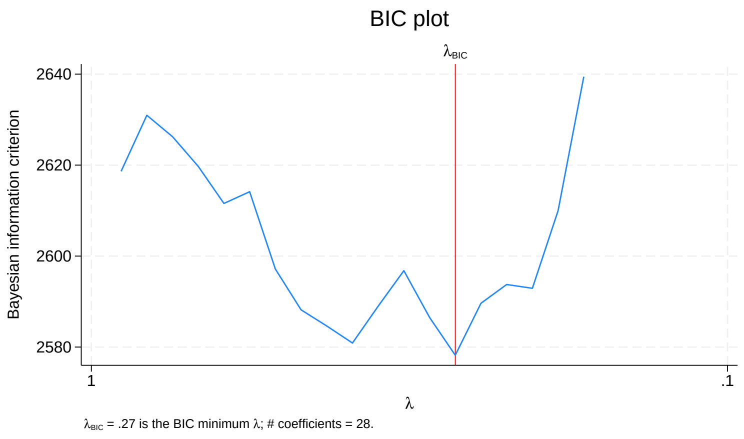
```
. lasso linear q104 ($idemographics) $ifactors $vlcontinuous
> if sample == 1, selection(bic)
Evaluating up to 100 lambdas in grid ...
Grid value 1:     lambda = .8978025   no. of nonzero coef. =   4
                  BIC = 2618.642
Grid value 2:     lambda = .8180442   no. of nonzero coef. =   7
                  BIC = 2630.961
Grid value 3:     lambda = .7453714   no. of nonzero coef. =   8
                  BIC = 2626.254
Grid value 4:     lambda = .6791547   no. of nonzero coef. =   9
                  BIC = 2619.727
Grid value 5:     lambda = .6188205   no. of nonzero coef. =  10
                  BIC = 2611.577
Grid value 6:     lambda = .5638462   no. of nonzero coef. =  13
                  BIC = 2614.155
Grid value 7:     lambda = .5137556   no. of nonzero coef. =  13
                  BIC = 2597.164
Grid value 8:     lambda =   .468115   no. of nonzero coef. =  14
                  BIC = 2588.189
Grid value 9:     lambda = .4265289   no. of nonzero coef. =  16
                  BIC = 2584.638
Grid value 10:    lambda = .3886373   no. of nonzero coef. =  18
                  BIC = 2580.891
Grid value 11:    lambda = .3541118   no. of nonzero coef. =  22
                  BIC = 2588.984
Grid value 12:    lambda = .3226535   no. of nonzero coef. =  26
                  BIC = 2596.792
Grid value 13:    lambda = .2939899   no. of nonzero coef. =  27
                  BIC = 2586.521
Grid value 14:    lambda = .2678726   no. of nonzero coef. =  28
                  BIC = 2578.211
Grid value 15:    lambda = .2440755   no. of nonzero coef. =  32
                  BIC = 2589.632
```

```
Grid value 16:     lambda = .2223925   no. of nonzero coef. =  35
                   BIC = 2593.753
Grid value 17:     lambda = .2026358   no. of nonzero coef. =  37
                   BIC = 2592.923
Grid value 18:     lambda = .1846342   no. of nonzero coef. =  42
                   BIC = 2609.975
Grid value 19:     lambda = .1682318   no. of nonzero coef. =  49
                   BIC = 2639.437
... selection BIC complete ... minimum found
Lasso linear model                           No. of obs        =        458
                                             No. of covariates =        277
Selection: Bayesian information criterion

--------------------------------------------------------------------------
         |                                No. of
         |                               nonzero    In-sample
      ID |     Description      lambda     coef.    R-squared          BIC
---------+----------------------------------------------------------------
       1 |    first lambda    .8978025         4       0.0308     2618.642
      13 |   lambda before    .2939899        27       0.3357     2586.521
    * 14 | selected lambda    .2678726        28       0.3563     2578.211
      15 |    lambda after    .2440755        32       0.3745     2589.632
      19 |     last lambda    .1682318        49       0.4445     2639.437
--------------------------------------------------------------------------
* lambda selected by Bayesian information criterion.
. estimates store linearbic
```

The minimum of the BIC function was found to be at $\lambda = 0.268$. It selects λ^* as this λ, which corresponds to 28 variables in the model out of 277 potential variables.

After fitting a lasso using BIC, it is a good idea to plot the BIC function and look at the shape of the curve around the minimum.

```
. bicplot
```



We see that the BIC function rises sharply once it hits the minimum. By default, the `lasso` command stops when it has identified a minimum.

So far, we have fit lasso linear models using CV, an adaptive lasso, and BIC. Which one performs better in the out-of-sample prediction?

```
. lassogof linearcv linearadaptive linearbic, over(sample) postselection
Postselection coefficients
─────────────────────────────────────────────────────────────
Name                sample │        MSE    R-squared        Obs
───────────────────────────┼─────────────────────────────────
linearcv                   │
                  Training │   8.652771       0.5065        503
                   Testing │   14.58354       0.2658        493
───────────────────────────┼─────────────────────────────────
linearadaptive             │
                  Training │   8.637575       0.5057        504
                   Testing │   14.70756       0.2595        494
───────────────────────────┼─────────────────────────────────
linearbic                  │
                  Training │   9.740229       0.4421        508
                   Testing │   13.44496       0.3168        503
─────────────────────────────────────────────────────────────
```

The BIC lasso performs the best.

More potential variables than observations

Lasso has no difficulty fitting models when the number of potential variables exceeds the number of observations.

We use `vl substitute` to create interactions of all of our factor-variable indicators with our continuous variables.

```
. vl substitute interact = i.factors##c.vlcontinuous
```

We fit the lasso.

```
. lasso linear q104 ($idemographics) $interact if sample == 1, rseed(1234)
note: 1.q32#c.q70 omitted because of collinearity with another variable.
note: 2.q34#c.q63 omitted because of collinearity with another variable.
  (output omitted)
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = 1.020288   no. of nonzero coef. =     4
Folds: 1...5....10   CVF = 16.93478
  (output omitted)
Grid value 34:    lambda = .2198144   no. of nonzero coef. =   106
Folds: 1...5....10   CVF = 12.91285
... cross-validation complete ... minimum found
Lasso linear model                          No. of obs        =        458
                                            No. of covariates =      7,227
Selection: Cross-validation                 No. of CV folds   =         10
─────────────────────────────────────────────────────────────────────────
         │                                 No. of    Out-of-      CV mean
         │                                nonzero     sample   prediction
      ID │     Description      lambda      coef.  R-squared        error
─────────┼───────────────────────────────────────────────────────────────
       1 │    first lambda    1.020288          4     0.0146     16.93478
      29 │   lambda before    .2773743         80     0.2531     12.83525
    * 30 │ selected lambda    .2647672         85     0.2545     12.81191
      31 │    lambda after    .2527331         89     0.2541     12.81893
      34 │     last lambda    .2198144        106     0.2486     12.91285
─────────────────────────────────────────────────────────────────────────
* lambda selected by cross-validation.
. estimates store big
```

There were 7,227 potential covariates in our model, of which lasso selected 85. That seems significantly more than the 49 selected by our earlier lasso.

Let's see how they do for out-of-sample prediction.

```
. lassogof linearcv big, over(sample) postselection
Postselection coefficients

─────────────────────────────┬───────────────────────────────────────────
Name                 sample  │         MSE    R-squared          Obs
─────────────────────────────┼───────────────────────────────────────────
linearcv                     │
                   Training  │    8.652771       0.5065          503
                    Testing  │    14.58354       0.2658          493
─────────────────────────────┼───────────────────────────────────────────
big                          │
                   Training  │    6.705183       0.6117          490
                    Testing  │    17.00972       0.1403          478
─────────────────────────────┴───────────────────────────────────────────
```

Our model with thousands of potential covariates did better for in-sample prediction but significantly worse for out-of-sample prediction.

Factor variables in lasso

It is important to understand how lasso handles factor variables. Let's say we have a variable, `region`, that has four categories representing four different regions of the country. Other Stata estimation commands handle factor variables by setting one of the categories to be the base level; it then makes indicator variables for the other three categories, and they become covariates for the estimation.

Lasso does not set a base level. It creates indicator variables for all levels (`1.region`, `2.region`, `3.region`, and `4.region`) and adds these to the set of potential covariates. The reason for this should be clear. What if `1.region` versus the other three categories is all that matters for prediction? Lasso would select `1.region` and not select the other three indicators. If, however, `1.region` was set as a base level and omitted from the set of potential covariates, then lasso would have to select `2.region`, `3.region`, and `4.region` to pick up the `1.region` effect. It might be wasting extra penalty on three coefficients when only one was needed.

See [LASSO] **Collinear covariates**.

Lasso logit and probit models

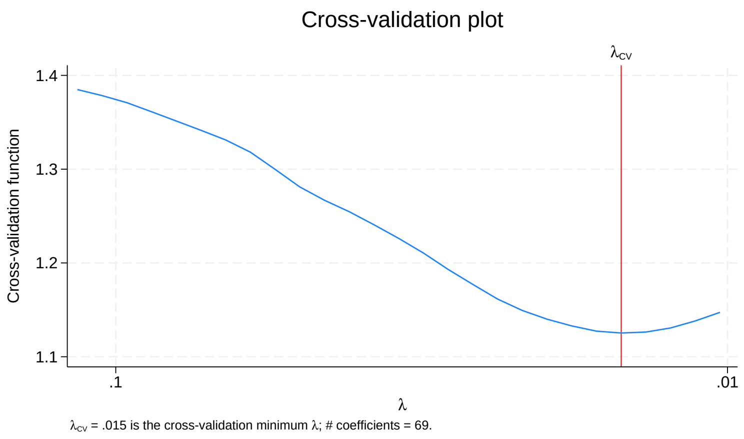
lasso will also fit logit, probit, Poisson, and Cox models.

We fit a logit model.

```
. lasso logit q106 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234)
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .1155342   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 1.384878
  (output omitted)
Grid value 27:    lambda =  .010285   no. of nonzero coef. =  88
Folds: 1...5....10   CVF = 1.147343
... cross-validation complete ... minimum found
Lasso logit model                           No. of obs         =        458
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds    =         10

--------------------------------------------------------------------------
         |                                No. of      Out-of-      CV mean
         |                               nonzero       sample
      ID |     Description      lambda     coef.   dev. ratio     deviance
---------+----------------------------------------------------------------
       1 |    first lambda    .1155342         0      -0.0004     1.384878
      22 |   lambda before    .0163767        65       0.1857     1.127315
    * 23 | selected lambda    .0149218        69       0.1871     1.125331
      24 |    lambda after    .0135962        73       0.1864     1.126333
      27 |     last lambda     .010285        88       0.1712     1.147343
--------------------------------------------------------------------------
* lambda selected by cross-validation.
. estimates store logit
```

Logit and probit lasso models are famous for having CV functions that are more wiggly than those for linear models.

```
. cvplot
```



This curve is not as smoothly convex as was the CV function for the linear lasso shown earlier. But it is not as bad as some logit CV functions. Because the CV functions for nonlinear models are not as smooth, `lasso` has a stricter criterion for declaring that a minimum of the CV function is found than it has for linear models. `lasso` requires that five smaller λ's to the right of a nominal minimum be observed with larger CV function values by a relative difference of `cvtolerance(`*#*`)` or more. Linear models only require three such λ's be found before declaring a minimum and stopping.

Let's now fit a probit model.

```
. lasso probit q106 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234)
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .1844415   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 1.384877
  (output omitted)
Grid value 26:    lambda = .0180201   no. of nonzero coef. =  87
Folds: 1...5....10   CVF = 1.152188
... cross-validation complete ... minimum found
Lasso probit model                         No. of obs        =        458
                                           No. of covariates =        277
Selection: Cross-validation                No. of CV folds   =         10

─────────────────────────────────────────────────────────────────────────
         │                                No. of      Out-of-
         │                               nonzero       sample      CV mean
      ID │     Description      lambda     coef.   dev. ratio     deviance
─────────┼───────────────────────────────────────────────────────────────
       1 │    first lambda    .1844415         0      -0.0004     1.384877
      21 │   lambda before    .0286931        61       0.1820     1.132461
    * 22 │ selected lambda    .0261441        64       0.1846     1.128895
      23 │    lambda after    .0238215        70       0.1841     1.129499
      26 │     last lambda    .0180201        87       0.1677     1.152188
─────────────────────────────────────────────────────────────────────────
* lambda selected by cross-validation.
. estimates store probit
```

`lassocoef` can be used to display coefficient values. Obviously, logit and probit coefficient values cannot be compared directly. But we do see similar relative scales.

```
. lassocoef logit probit, sort(coef, standardized) display(coef, standardized)

                           
                |     logit     probit
----------------+----------------------
           q142 |
             No |    -.50418  -.3065817
                |
           q154 |
             No |  -.3875702  -.2344515
                |
            q90 |
             No |  -.3771052  -.2288992
                |
             q8 |
             No |  -.3263827   -.200673
                |
  (output omitted)
                |
            q37 |
             No |  -.0128537  -.0062874
         2.q158 |   .0065661   .0012856
          3.q65 |  -.0062113
         3.q110 |  -.0055616
           q120 |   .0044864
         0.q146 |   -.004312
                |
            q95 |
              3 |   .0030261
---------------------------------------
Legend:
  b - base level
  e - empty cell
  o - omitted
```

The probit lasso selected five fewer variables than logit, and they were the five variables with the smallest absolute values of standardized coefficients.

We look at how they did for out-of-sample prediction.

```
. lassogof logit probit, over(sample)
Penalized coefficients

                              |               Deviance
Name                   sample |   Deviance      ratio        Obs
------------------------------+----------------------------------
logit                         |
                     Training |   .8768969     0.3674        499
                      Testing |   1.268346     0.0844        502
------------------------------+----------------------------------
probit                        |
                     Training |   .8833892     0.3627        500
                      Testing |    1.27267     0.0812        503
-----------------------------------------------------------------
```

Neither did very well. The out-of-sample deviance ratios were notably worse than the in-sample values. The deviance ratio for nonlinear models is analogous to R^2 for linear models. See *Methods and formulas* for [LASSO] **lassogof** for the formal definition.

We did not specify the `postselection` option in this case because there are no theoretical grounds for using postselection coefficients for prediction with nonlinear models.

Lasso Poisson models

Next, we fit a Poisson model.

```
. lasso poisson q107 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234)

10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .5745539   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 2.049149
  (output omitted)
Grid value 21:    lambda =  .089382   no. of nonzero coef. =  66
Folds: 1...5....10   CVF = 1.653376
... cross-validation complete ... minimum found

Lasso Poisson model                          No. of obs         =        458
                                             No. of covariates  =        277
Selection: Cross-validation                  No. of CV folds    =         10

--------------------------------------------------------------------------
         |                                No. of      Out-of-
         |                               nonzero       sample      CV mean
      ID |     Description      lambda     coef.   dev. ratio     deviance
---------+----------------------------------------------------------------
       1 |    first lambda    .5745539         0      -0.0069     2.049149
      16 |   lambda before    .1423214        37       0.1995     1.629222
    * 17 | selected lambda     .129678        45       0.1999     1.628315
      18 |    lambda after    .1181577        48       0.1993      1.62962
      21 |     last lambda     .089382        66       0.1876     1.653376
--------------------------------------------------------------------------
* lambda selected by cross-validation.
```

We see how it does for out-of-sample prediction.

```
. lassogof, over(sample)

Penalized coefficients

-----------------------------------------------------
             |                 Deviance
      sample |   Deviance         ratio          Obs
-------------+---------------------------------------
    Training |   1.289175        0.3515          510
     Testing |   1.547816        0.2480          502
-----------------------------------------------------
```

Its in-sample and out-of-sample predictions are fairly close. Much closer than they were for the logit and probit models.

Lasso Cox models

`lasso` will also fit Cox proportional hazards models. We illustrate `lasso cox` with an example that predicts risk of death for stage I lung adenocarcinoma patients. Lung adenocarcinoma is one of the most common non-small-cell lung cancers.

Stage I adenocarcinoma indicates that the tumor size is relatively small, and cancer has not spread to other distant organs. Stage I adenocarcinoma patients usually have varied survival outcomes even though they are in the early cancer development stage. For example, Yu et al. (2016) show that, in one cohort, more than 50% of stage I adenocarcinoma patients died within 5 years after the initial diagnosis, while about 15% of the patients survived for more than 10 years.

Histopathology image features are indispensable for prognostic analysis. Examples of the histopathology image features include image granularity, image intensity, cell size and shape, pixel intensity of the cell, cell texture, area occupied by cells, neighboring relation of the cells, nucleus size and shape, and nucleus texture. We can use `lasso cox` to extract the top histopathology image features that distinguish short-term survivors from long-term survivors.

We have a fictitious survival dataset (`lungcancer.dta`) inspired by Yu et al. (2016). The variable `t` records either the time of death or censoring in months for stage I adenocarcinoma lung cancer patients. The indicator variable `died` is 1 or 0 if the patient died or is censored, respectively. There are 500 histopathology image features, `histfeature1` to `hisfeature500`, and only 250 patients. The analysis aims to classify a new patient into a low-risk or high-risk group, given the histopathology image features.

We first load the dataset and then type `stset` to show it has already been `stset`.

```
. use https://www.stata-press.com/data/r19/lungcancer
(Fictitious data on stage I adenocarcinoma lung cancer)

. stset
-> stset t, failure(died)

Survival-time data settings

         Failure event: died!=0 & died<.
Observed time interval: (0, t]
     Exit on or before: failure

--------------------------------------------------------------------------

        250  total observations
          0  exclusions
--------------------------------------------------------------------------

        250  observations remaining, representing
        211  failures in single-record/single-failure data
 18,465.093  total analysis time at risk and under observation
                                                At risk from t =         0
                                     Earliest observed entry t =         0
                                          Last observed exit t =       260
```

Next, we need to split the entire sample into training and testing data. The training data will be used for estimation, and the testing data will be used to measure the prediction performance. These steps are typically used in the microarray survival literature; for an application to the performance of a Cox model with lasso, see Sohn et al. (2009).

We use `splitsample` to split the data into two parts. The `generate(group)` option creates a new variable `group` for the identification of the training and testing data. That is, `group` equals 1 if it belongs to the training data or 0 if it belongs to the testing data. The `split(0.6 0.4)` option specifies that 60% of the entire data be used as training data and 40% of them be used as testing data. To make the results reproducible, we specify the `rseed()` option.

```
. splitsample, generate(group) split(0.6 0.4) rseed(12345)
```

For the convenience of later use, we separately save the training data (`lungcancer_training.dta`) and the testing data (`lungcancer_testing.dta`).

```
. preserve
. keep if group == 1
(100 observations deleted)
. save lungcancer_training
file lungcancer_training.dta saved
. restore
. preserve
. keep if group == 2
(150 observations deleted)
. save lungcancer_testing
file lungcancer_testing.dta saved
. restore
```

We are now ready to fit a `lasso cox` model using only the training data. By default, we use cross-validation. We specify `rseed()` to make the results reproducible.

```
. use lungcancer_training, clear
(Fictitious data on stage I adenocarcinoma lung cancer)
. lasso cox histfeature*, rseed(12345671)
        Failure _d: died
  Analysis time _t: t
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .3539123   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 8.922501
Grid value 2:     lambda = .3378265   no. of nonzero coef. =   1
Folds: 1...5....10   CVF = 8.917438
  (output omitted)
Grid value 30:    lambda = .0918411   no. of nonzero coef. =  45
Folds: 1...5....10   CVF = 8.042941
Grid value 31:    lambda = .0876668   no. of nonzero coef. =  48
Folds: 1...5....10   CVF = 8.039609
Grid value 32:    lambda = .0836822   no. of nonzero coef. =  52
Folds: 1...5....10   CVF =  8.05246
Grid value 33:    lambda = .0798787   no. of nonzero coef. =  57
Folds: 1...5....10   CVF = 8.070293
Grid value 34:    lambda = .0762481   no. of nonzero coef. =  63
Folds: 1...5....10   CVF = 8.105045
... cross-validation complete ... minimum found
Lasso Cox model                            No. of obs        =        150
                                           No. of covariates =        500
Selection: Cross-validation                No. of CV folds   =         10

---------------------------------------------------------------------------
         |                                No. of
         |                               nonzero    In-sample      CV mean
      ID |     Description      lambda     coef.   dev. ratio     deviance
---------+-----------------------------------------------------------------
       1 |    first lambda    .3539123         0       0.0000     8.922501
      30 |   lambda before    .0918411        45       0.2199     8.042941
    * 31 | selected lambda    .0876668        48       0.2306     8.039609
      32 |    lambda after    .0836822        52       0.2419      8.05246
      34 |     last lambda    .0762481        63       0.2662     8.105045
---------------------------------------------------------------------------
* lambda selected by cross-validation.
```

lasso cox selects 48 of the 500 features. We can now predict the relative-hazard ratio, which we will call riskscore_training, and evaluate risk scores. We will use the median of riskscore_training as a threshold to classify a patient as low risk or high risk. We store the median value in a global macro (median) for later use.

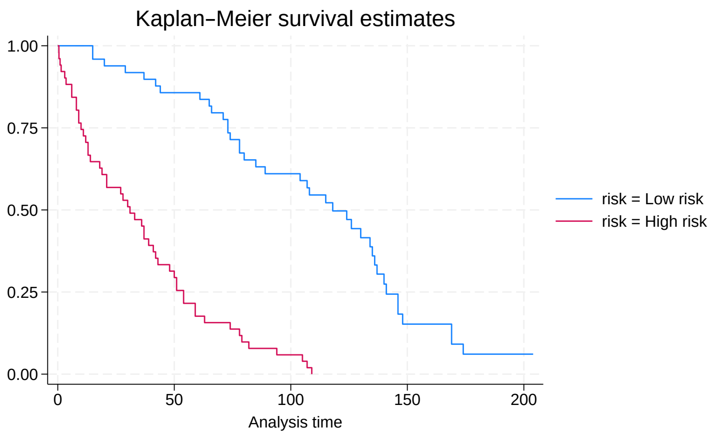
```
. predict riskscore_training
(options hr penalized assumed; predicted hazard ratio with penalized
coefficients)

. summarize riskscore_training, detail

              Predicted hazard ratio, penalized
-------------------------------------------------------------

      Percentiles      Smallest
 1%      .054982       .0414753
 5%     .0838301        .054982
10%     .1308778       .0702972       Obs                 150
25%     .3676802       .0727958       Sum of wgt.         150

50%     .9458244                      Mean           1.998198
                        Largest       Std. dev.       3.75226
75%     2.368032       9.962103
90%     4.912702       11.13334       Variance       14.07945
95%     6.651043        12.4411       Skewness       7.054249
99%      12.4411       39.40631       Kurtosis       67.68195

. global median = r(p50)
```

Based on the median of the predicted risk ratio in the training data, we now use the testing data to validate the model. First, we predict the risk ratio in the testing sample, which we will call riskscore_testing. Then, we compare riskscore_testing with the median of the risk ratio obtained in the training data ($median). If the predicted risk score is greater than or equal to the median, the patient is labeled as high risk. If the predicted risk score is less than the median, the patient is classified as low risk.

```
. use lungcancer_testing, clear
(Fictitious data on stage I adenocarcinoma lung cancer)

. predict riskscore_testing
(options hr penalized assumed; predicted hazard ratio with penalized
coefficients)

. generate byte risk = (riskscore_testing >= $median)

. label define risk_lb 1 "High risk" 0 "Low risk"

. label values risk risk_lb
```

To evaluate the effectiveness of risk classification, we first look at the Kaplan–Meier plot, which draws the survival curve for both low-risk and high-risk groups.

```
. sts graph, by(risk)
        Failure _d: died
  Analysis time _t: t
```



The graph shows that the predicted high-risk patients have a more steeply falling survival curve than the predicted low-risk patients. To confirm this conjecture, we do a log-rank test.

```
. sts test risk
        Failure _d: died
  Analysis time _t: t

Equality of survivor functions
Log-rank test

          │  Observed       Expected
risk      │    events         events
──────────┼─────────────────────────
 Low risk │        39          68.17
High risk │        51          21.83
──────────┼─────────────────────────
    Total │        90          90.00

                    chi2(1) =   61.50
                    Pr>chi2 = 0.0000
```

The log-rank test rejects the hypothesis that the predicted low-risk and high-risk patients have the same survival functions. Both the Kaplan–Meier plot and the log-rank test show that using the predicted hazard ratios' median can effectively distinguish a low-risk patient from a high-risk patient. We can now make prognostic predictions given new data.

The dataset (`newlungcancer.dta`) contains histopathology image features for some new stage I adenocarcinoma patients, but their survival time is not recorded because they are still alive. Based on the prediction model from `lasso cox`, we want to classify these new patients as low risk or high risk. To achieve this objective, we need to predict the new patients' hazard ratios and compare them with the median level of risk score obtained in the training data.

```
. use https://www.stata-press.com/data/r19/newlungcancer, clear
(Fictitious new data on stage I adenocarcinoma lung cancer)

. predict riskscore_new
(options hr penalized assumed; predicted hazard ratio with penalized
coefficients)

. generate risk = (riskscore_new >= $median)

. label define risk_lb 1 "High risk" 0 "Low risk"

. label values risk risk_lb

. tabulate risk

       risk |      Freq.     Percent        Cum.
------------+-----------------------------------
   Low risk |         27       54.00       54.00
  High risk |         23       46.00      100.00
------------+-----------------------------------
      Total |         50      100.00
```

The table of the predicted risk level shows that 27 patients are classified as low risk, while 23 patients are classified as high risk.

References

Sohn, I., J. Kim, S.-H. Jung, and C. Park. 2009. Gradient lasso for Cox proportional hazards model. *Bioinformatics* 25: 1775–1781. https://doi.org/10.1093/bioinformatics/btp322.

Yu, K., C. Zhang, G. J. Berry, R. B. Altman, C. Ré, D. L. Rubin, and M. Snyder. 2016. Predicting non-small cell lung cancer prognosis by fully automated microscopic pathology image features. *Nature Communications* 7(12474). https://doi.org/10.1038/ncomms12474.

Also see

[LASSO] **lasso** — Lasso for prediction and model selection

[LASSO] **lasso fitting** — The process (in a nutshell) of fitting lasso models

lasso fitting — The process (in a nutshell) of fitting lasso models

Description Remarks and examples Also see

Description

This entry describes the process of fitting lasso models.

Remarks and examples

Remarks are presented under the following headings:

Introduction

If you are to fit lasso models successfully, you need to understand how the software computes them. There are options you can specify to modify the process and specifying them is sometimes necessary just to find the solution. This entry explains the process.

The process of fitting lasso models applies to the three commands that directly fit lasso and related models:

`lasso`	`sqrtlasso`	`elasticnet`

The lasso inferential commands

`dsregress`	`poregress`	`xporegress`
`dslogit`	`pologit`	`xpologit`
`dspoisson`	`popoisson`	`xpopoisson`
	`poivregress`	`xpoivregress`

fit multiple lasso models under the hood, and you may want to try using different lasso model-selection methods with these commands. If you do, then this entry is also for you. All the options described here can be used with the inferential commands to specify different lasso model-selection methods and modify the settings that control the lasso-fitting process.

Model selection

Fitting lasso models requires that the software fit lots of models behind the scenes from which one will be selected. The trick to minimizing the time needed is to avoid fitting unnecessary models, while ensuring that you fit the right one so it is there to select.

Lasso has a way of ordering models. They are ordered on scalar parameter λ defined over 0 to $+\infty$. λ is a parameter of the penalty function. For ordinary lassos, the penalty is λ times the sum of the absolute values of the coefficients of the normalized variables. Every possible model has a λ associated with it. When λ is large, the penalty is large, and the model has few or no variables. Models with smaller λ's have more variables.

We can think of lasso as fitting model(λ), where λ varies over a range, and then selecting one of them.

Which model do we select? That depends on the problem. Do we want a good model for prediction or a parsimonious model that better reflects the "true" model?

One method of selection is called cross-validation (CV), and it works well for prediction. The criterion is the CV function $f(\lambda)$, an estimate of the out-of-sample prediction error, which we minimize. The model for the λ that minimizes the CV function is the selected model.

To find λ^* that minimizes $f(\cdot)$, and thus find the corresponding model(λ^*), we need to fit models with λ's near to λ^* to be certain that we identified the minimum. Only nearby λ's would be good enough if your models were fit on infinite-observation datasets because this perfect $f(\lambda)$ is globally convex. Because your datasets will be finite, the empirically estimated function will jiggle around its Platonic ideal, and that means that you will need to fit additional models to be reasonably certain that the one you select is the one that corresponds to λ^*.

Another method, adaptive lasso, works well when the goal is to find parsimonious models—models with fewer variables in them—that might better reflect the true model. Adaptive lasso starts by finding the CV solution and then, using weights on the coefficients in the penalty function, does another lasso and selects a model that has fewer variables.

A third method is called plugin lasso. It is faster than CV or adaptive lasso. It is not just faster, it will be approaching the finish line while the others are still working out where the finish line is. It is faster because it does not minimize $f(\cdot)$. It uses an iterative formula to calculate the smallest λ that is large enough to dominate the estimation error in the coefficients. Plugin will not produce as low an out-of-sample prediction error as CV, but it will produce more parsimonious models than CV. Plugin is the default selection method for the inferential commands because it is so fast. For prediction, CV has better theoretical properties.

A fourth method uses the Bayesian information criterion (BIC) function to select λ^*. That is, $f(\lambda)$ is the BIC function, and λ^* minimizes $f(\cdot)$. The number of covariates selected by minimizing BIC typically lies between the number selected by CV and the number selected by the plugin method; however, BIC tends to be more similar to the number selected by the plugin method. Furthermore, BIC does not require a complex derivation as does the plugin, so like CV, it can be applied in a more general context. Typically, selection using BIC is much faster than selection using CV, but this is not always the case.

We discuss CV, adaptive, plugin, and BIC lassos below, and we discuss a fifth selection method that we call none. None is a do-it-yourself (DIY) method. It calculates model(λ) over a range of λ's and stops. You then examine them and choose one.

The process

Step 1. Set the grid range

Step 1 consumes virtually no time, but the total time steps 2 and 3 consume will depend on the grid that step 1 sets. The grid that steps 2 and 3 will search and calculate over will range from λ_{gmax} to λ_{gmin} and have G points on it.

Large values of λ correspond to models with few or no variables in them. Small values correspond to models with lots of variables. Given any two values of λ, λ_1, and λ_2,

$$\lambda_1 > \lambda_2 \text{ usually implies that \# of variables in model } 1 \leq \text{\# of variables in model } 2$$

Most of us think of parameters as running from smallest to largest, say, 0 to $+\infty$, but with λ, you will be better served if you think of them as running from $+\infty$ to 0.

The grid does not start at $+\infty$, it starts at λ_{gmax}. The software does an excellent job of setting λ_{gmax}. It sets λ_{gmax} to the smallest λ that puts no variables in the model. You cannot improve on this. There is no option for resetting λ_{gmax}.

The software does a poor job of setting λ_{gmin}. There simply does not exist a scheme to set it optimally. If we are to identify the minimum of the CV function, $f(\lambda^*)$, λ_{gmin} must be less than λ^*. That is difficult to do because obviously we do not know the value of λ^*.

Computing models for small λ's is computationally expensive because fitting a model for a small λ takes longer than fitting a model for a larger λ. Our strategy is to hope we set λ_{gmin} small enough and then stop iterating over λ as soon as we are assured that we have found the minimum of the CV function. If we did not set λ_{gmin} small enough, the software will tell us this.

The initial grid is set to

$$\lambda_{\text{gmax}},\ \lambda_2,\ \lambda_3,\ \ldots,\ \lambda_{\text{gmin}} \quad (\lambda_{\text{gmin}} \text{ too small we hope})$$

The software sets λ_{gmin} to *ratio* $\times\ \lambda_{\text{gmax}}$, where *ratio* defaults to 1e–4 when $p < N$, where p is the number of potential covariates and N the number of observations. When $p \geq N$, the default is 1e–2.

You can reset *ratio* with the `grid(, ratio(#))` option, or you can reset λ_{gmin} directly by specifying `grid(, min(#))`.

Finally, in addition to setting *ratio* or λ_{gmin}, you can reset the number of points on the grid. It is set to 100 by default, meaning the initial grid will be

$$\lambda_{\text{gmin}} = \lambda_1,\ \lambda_2,\ \lambda_3,\ \ldots,\ \lambda_{99},\ \lambda_{\text{gmin}} = \lambda_{100}$$

You can reset the number of points by specifying `grid(#)`. You can specify the number of points and a value for *ratio* by typing `grid(#, ratio(#))`. See [LASSO] **lasso**.

Step 2. Fit the model for next lambda in grid

We have a grid range λ_{gmax} to λ_{gmin} and number of points on the grid, which we will simply denote by their indices:

$$\lambda_1,\ \lambda_2,\ \lambda_3,\ \ldots,\ \lambda_{99},\ \lambda_{100}$$

The software obtains the models

$$\text{model}(\lambda_1),\ \text{model}(\lambda_2),\ \text{model}(\lambda_3),\ \ldots,\ \text{model}(\lambda_{100})$$

By “obtains”, we mean that the software chooses the variables that appear in each one. The software proceeds from left to right. The first model, model(λ_1), has no variables in it and was easy to find. Once found, model(λ_1) provides the starting point for finding model(λ_2), and model(λ_2) provides the starting point for finding model(λ_3), and so on. Working from the previous model to obtain the next model is known as a warm start in the literature. Regardless of what the technique is called, this is why the software does not allow you to set a different λ_{gmax} for λ_1. To calculate model(λ) for a small value of λ , the software has to work its way there from previous model(λ) results.

The grid points are not equally spaced. The grid points are not

$$\begin{aligned}
\lambda_1 &= \lambda_{\text{gmax}} \\
\lambda_2 &= \lambda_1 - \Delta \\
\lambda_3 &= \lambda_2 - \Delta \\
\lambda_4 &= \lambda_3 - \Delta \\
&\vdots
\end{aligned}$$

The grid points are instead chosen so that $\ln \lambda$ is equally spaced, which you can think of as the λ’s being closer together as they get smaller:

$$\begin{aligned}
\lambda_1 &= \lambda_{\text{gmax}} \\
\lambda_2 &= \lambda_1 - \Delta_1 \\
\lambda_3 &= \lambda_2 - \Delta_2, \quad 0 < \Delta_2 < \Delta_1 \\
\lambda_4 &= \lambda_3 - \Delta_3, \quad 0 < \Delta_3 < \Delta_2 \\
&\vdots
\end{aligned}$$

Model estimation involves not only choosing the variables that appear in the model but also estimating their coefficients as well.

The computation will not usually be carried out all the way to λ_{100}. Because small λ’s are computationally expensive, we want to stop before we get to λ_{100}. There are two criteria for stopping. The first is when we have identified the minimum of the CV function.

After we fit model(λ_1), we compute the value of the CV function for λ_1, $f(\lambda_1)$. Likewise after fitting model(λ_2), we compute $f(\lambda_2)$. For early λ’s, typically we have $f(\lambda_k) > f(\lambda_{k+1})$. Now if we see

$$f(\lambda_{k-1}) > f(\lambda_k) < f(\lambda_{k+1})$$

λ_k might give the minimum of the CV function. It is possible that the CV function is bouncing around a bit, and it might not be the true minimum. We discuss how we declare a minimum in more detail in the next section.

For now, assume that we have properly identified a minimum. We are done, and we need not do any more estimations of model(λ).

But what if we do not find a minimum of the CV function? Sometimes, the CV function flattens out and stays flat, barely changing and only slowly declining with each smaller λ.

As the software proceeds from the calculation of model(λ_{k-1}) to model(λ_k), it calculates the relative differences in the in-sample deviances between models:

$$\frac{\text{deviance}\{\text{model}(\lambda_{k-1})\} - \text{deviance}\{\text{model}(\lambda_k)\}}{\text{deviance}\{\text{model}(\lambda_{k-1})\}}$$

This relative difference is a measure of how much predictive ability is added by proceeding to model(λ_k). If it is small, that suggests the difference between the CV function values $f(\lambda_{k-1})$ and $f(\lambda_k)$ will be small, and changes in the function for smaller λ's smaller yet. So we think it is likely that we have gone far enough.

If the relative difference is less than 1e–5, the software sets the selected $\lambda^* = \lambda_{\text{stop}} = \lambda_k$ and stops estimating models for more λ's. The output tells you that the selected λ^* was determined by this stopping rule. This means model(λ^*) does not give the minimum of the CV function, but we believe something close to it.

If you do not want this default behavior, there are three things you can do. The first is to change the value of the stopping rule tolerance. If you want to use 1e–6 instead of 1e–5, specify

```
. lasso y x1 x2 ..., stop(1e-6)
```

With a smaller tolerance, it will iterate over more λ's, giving a greater chance that a minimum might be identified.

The second possibility is to turn off the early stopping rule by setting the tolerance to 0. If there is a minimum that can be identified, this will find it.

```
. lasso y x1 x2 ..., stop(0)
```

If, however, the CV function flattens out and stays flat, specifying `stop(0)` might mean that the software iterates to the end of the λ grid, and this might take a long time.

A third choice is to specify

```
. lasso y x1 x2 ..., selection(cv, strict)
```

This is the same as the default behavior in this case, except that it throws an error! The suboption `strict` says that if we do not find a minimum, end with an error. This is useful when using `selection(cv)` with the inferential commands. It alerts us to the fact that we did not find a minimum, and it leaves the lasso behind, so we can plot the CV function and decide what to do next.

Selection method none

If you specify `selection(none)` instead of `selection(cv)`, the software stops when the stopping rule tolerance is reached or when the end of the λ grid is reached.

You can specify `selection(none)` when you want to gain a feel for how the number of included variables changes over λ or if you want to choose λ^* yourself. We provide a suite of postestimation commands for this purpose:

- `lassoknots` shows you a table of the λ's and the properties of the models.
- `lassocoef` lists the variables in the selected model. It can compare multiple models in the same table.

- `lassoselect` lets you choose a model to be treated as the selected model(λ^*).
- `lassogof` evaluates the selected model. It can also compare multiple models in the same table.

What you do not have, however, is the CV function and other CV-based measures of fit, which allow you to evaluate how well models predict and so make an informed choice as to which model should be model(λ^*).

There is another way. Do not specify `selection(none)`, specify `selection(cv)` or `selection(adaptive)`. The above postestimation functions will work, and you can, based on your own criteria if you wish, select the model for yourself.

Step 3. Identifying a minimum of the CV function

The minimum is identified when there are values of $f(\cdot)$ that rise above it on both sides. For example, consider the following case:

$$f(\lambda_1) > f(\lambda_2) > \cdots > f(\lambda_{49})$$

and

$$f(\lambda_{49}) < f(\lambda_{50}) < f(\lambda_{51}) < f(\lambda_{52})$$

For linear models, $f(\lambda_{49})$ is an identified minimum, and the software sets $\lambda^* = \lambda_{49}$. Linear models require that there be three smaller λ's with larger CV function values by a relative difference of `cvtolerance(#)` or more.

Because the CV functions for nonlinear models are not as smooth, `lasso` has a stricter criterion for declaring that a minimum of the CV function is found than it has for linear models. `lasso` requires that five smaller λ's to the right of a nominal minimum be observed with larger CV function values by a relative difference of `cvtolerance(#)` or more.

If you want more assurance that you have found a minimum, you can change `cvtolerance(#)` to a larger value from its default of 1e–3.

```
. lasso y x1 x2 ..., cvtolerance(1e-2)
```

Making the tolerance larger typically means that a few more model(λ)'s are estimated to find the required three (or five) with CV function values larger than the minimum by this tolerance.

The software provides three options that control how λ^* is set when a identified minimum is not found. They work like this:

	λ^* is set to		
Options	Case 1	Case 2	Case 3
`selection(cv, strict)`	λ_{cvmin}	error	error
`selection(cv, stopok)`	λ_{cvmin}	λ_{stop}	error
`selection(cv, gridminok)`	λ_{cvmin}	λ_{stop}	λ_{gmin}

Case 1 is an identified minimum.
Case 2 is falling over range, stopping rule tolerance reached.
Case 3 is falling over range, stopping rule tolerance not reached.

λ_{cvmin} is the identified minimum of the CV function $f(\cdot)$.
λ_{stop} is the λ that meant the stopping rule tolerance.
λ_{gmin} is the last λ in the grid.
error indicates that λ^* is not set, and the software issues an error message.

You may specify only one of the three options. `selection(cv, stopok)` is the default if you do not specify one.

We emphasize that these options affect the setting of λ^* only when an identified minimum is not found.

`selection(cv, stopok)` is the default and selects $\lambda^* = \lambda_{\text{stop}}$ when the stopping rule tolerance was reached.

`selection(cv, strict)` is the purist's option. λ^* is found only when a minimum is identified. Otherwise, the software issues a minimum-not-found error.

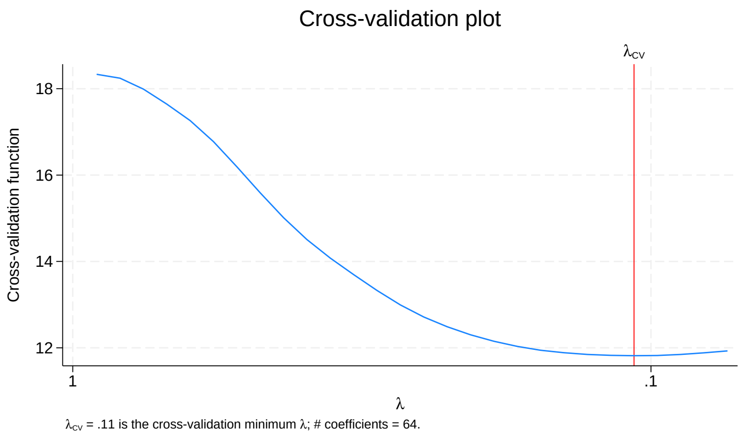
`selection(cv, gridminok)` is an option that has an effect only when the early stopping rule tolerance is not reached. We have fallen off the right edge of the grid without finding an identified minimum. λ^* is set to λ_{gmin}. There is no theoretical justification for this rule. Practically, it means that λ_{gmin} was set too large. We should make it smaller and refit the model.

Plotting the CV function

Run `lasso` if you have not already done so. After you do, there are two possible outcomes. The software ended by setting a λ^*, thus selecting a model, or it did not set a λ^*. You will have no doubts as to which occurred because when λ^* is not set, the software ends with an error message and a nonzero return code. Note that even when it ends with a nonzero return code, results of the lasso are left behind.

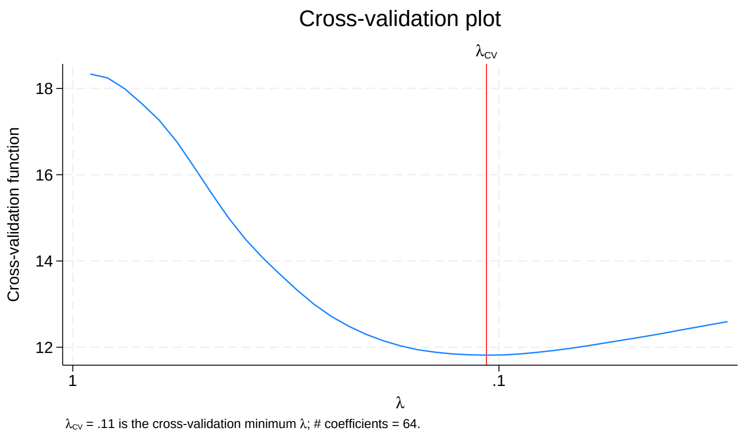
Regardless of how estimation ended, graph the CV function, $f(\cdot)$. It is easy to do. Type `cvplot` after running a lasso. Here is one:

```
. lasso linear y x1 x2 ...
. cvplot
```



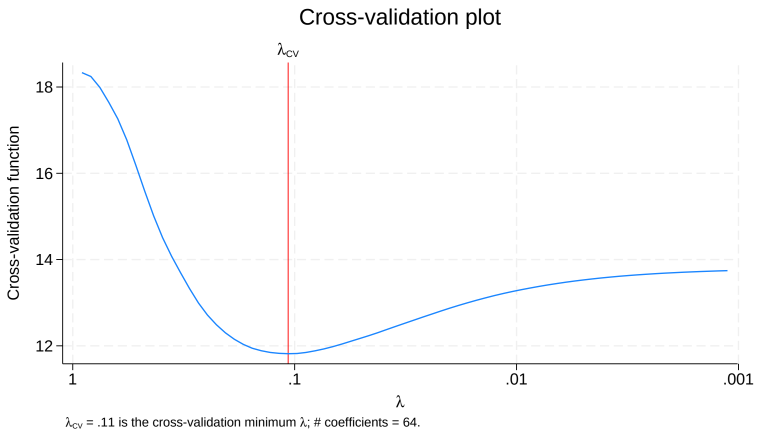
This lasso identified a minimum of the CV function. It identified the minimum and stopped iterating over λ. If we want to see more of the CV function, we can set `cvtolerance(#)` to a larger value.

```
. lasso linear y x1 x2 ..., cvtolerance(0.05)
. cvplot
```



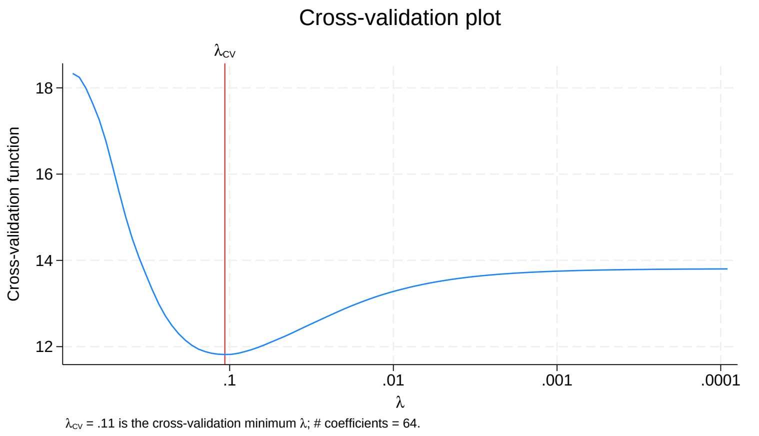
If we want to see more of the CV function, we can specify `selection(cv, alllambdas)`. When the `alllambdas` suboption is specified, estimation does not end when a minimum of the CV function is found. In fact, it estimates model(λ) for all λ's first and then computes the CV function because this is slightly more computationally efficient if we are not stopping after identifying a minimum.

```
. lasso linear y x1 x2 ..., selection(cv, alllambdas)
. cvplot
```



Actually, `alllambdas` is a lie. In this case, it estimated only 73 λ's. It ended when the stopping rule tolerance was reached. If we really want to see all 100 λ's, we need to turn off the stopping rule.

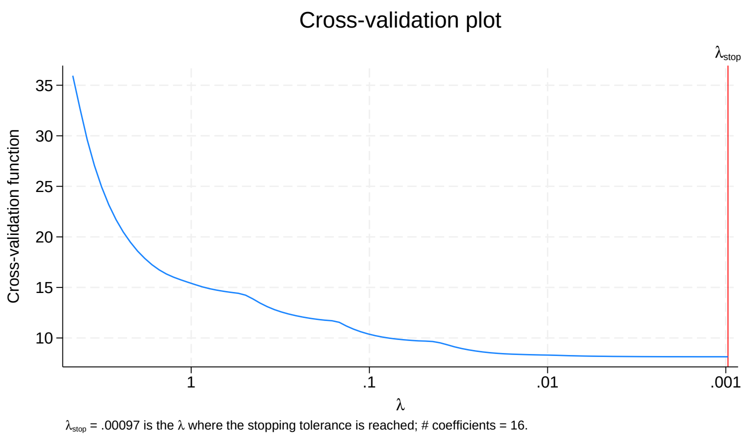
```
. lasso linear y x1 x2 ..., selection(cv, alllambdas) stop(0)
. cvplot
```



That is a plot of all 100 λ's. Clearly, in this case, the default behavior worked fine to identify a minimum.

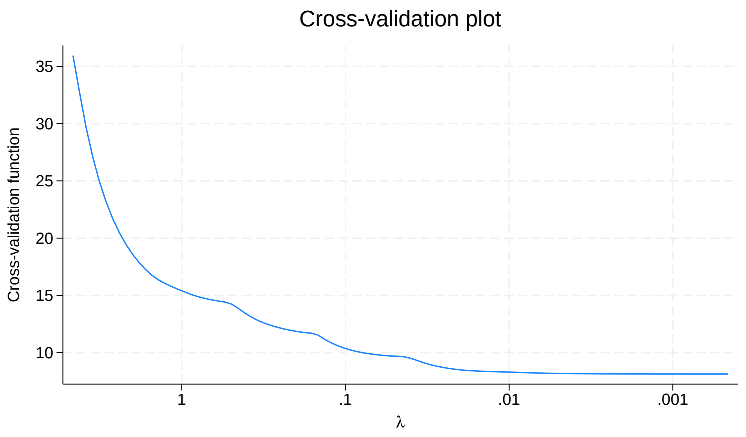
Here is an example of a CV function for which a minimum was not identified. The stopping rule tolerance was reached instead.

```
. lasso linear z w1 w2 ...
. cvplot
```



To try more λ's in a search for a minimum, we turn off the stopping rule

```
. lasso linear z w1 w2 ..., stop(0)
. cvplot
```



It went to the end of the grid without finding a minimum. The default stopping rule tolerance usually works fine. Setting `stop(0)` typically burns more computer time without identifying a minimum.

Selecting another model

Imagine that you have successfully found the model that minimizes the CV function, $f(\lambda)$, the estimate of out-of-sample prediction error. If your interest is in prediction, the model that minimizes the CV function really is best. If your interest is in model selection, however, you may want to look at alternatives that are close in the out-of-sample prediction sense.

You can use `lassoknots` to see a table of the λ's where variables were added or dropped. These are called knot points.

You can then use `lassoselect` to choose one of the models. This command sets λ^* to the λ you specify. Once you have selected a model, you can use all of `lasso`'s postestimation features on it. And then, if you wish, you can `lassoselect` another model. If you use `estimates store` after each `lassoselect`, you can compare multiple models side by side using `lassogof`.

See [LASSO] **lassoselect** for an example.

What exactly is CV?

We are done discussing using CV as a selection method, and yet we have never discussed CV itself. CV is about using one subsample of data to fit models and another to evaluate their prediction error.

Here are the details. The $f(\cdot)$ function is an estimate of the out-of-sample prediction error, and the function is calculated using CV. The method starts by dividing the data into K partitions called folds. Once that is done, for each fold k,

1. model(λ) is fit on all observations except those in fold k.
2. that result is used to predict the outcome in fold k.
3. steps 1 and 2 are repeated for each fold.
4. the prediction error is then averaged over all folds, which is to say, all observations. This is $f(\lambda)$.

Option `selection(cv, folds(#))` sets K, and `folds(10)` is used by default.

Adaptive lasso

In *Plotting the CV function*, we looked at a graph of the CV function for which $f(\lambda)$ had a long flat region and the stopping rule selected λ^*. We explained that you could use the `lasso` DIY postestimation commands to change the selected model to one with fewer variables in it.

Adaptive lasso is another approach for obtaining parsimonious models. It is a variation on CV, and in fact, for each step, it uses CV. It uses the CV-selected model(λ^*) as a starting point and then amplifies the important coefficients and attenuates the unimportant ones in one or more subsequent lassos that also use CV.

For the second lasso, variables not selected in the first lasso's model(λ^*) are dropped, and the penalty term uses weights equal to the inverse of the absolute value of the coefficients from model(λ^*). The justification being that important coefficients are large and unimportant ones, small. (Variables are standardized so that comparison of coefficient size makes sense.) These weights tend to drive small coefficients to zero in the second lasso. So the selected model from the second lasso almost always has fewer variables than the selected model from the first lasso.

Plugin selection

CV selects model(λ^*) such that $f(\lambda)$ is minimized. Adaptive is a variation on CV. It selects a final model(λ^*) that minimizes a more restricted $f(\lambda)$.

Plugins—`selection(plugin)`—are a whole different thing. Parameter λ still plays a role, but $f(\cdot)$ does not. Instead, the λ^* that determines model(λ^*) is produced by direct calculation using the plugin function, $\lambda^* = g(\cdot)$. The function returns the smallest value of λ that is large enough to dominate the estimation error in the coefficients.

No search over λ is required, nor is a grid necessary. This makes plugin the fastest of the methods provided. It is fast, but it is not instantaneous. The plugin formula is solved iteratively, and if it is trying to calculate a small value for λ^*, it can take a little time. Those small λ's again!

Plugin's selected model(λ^*) are almost always more parsimonious than the minimum-$f(\lambda)$ models selected by CV. Plugin will not produce models with as low an out-of-sample prediction error as CV, but it tends to select the most important variables and can be proven to do so for many data-generation processes. Plugin is popular when the problem is model selection instead of out-of-sample prediction.

Selection using the BIC function

Selecting λ^* using the BIC function—`selection(bic)`—is similar to selection using CV. However, rather than the CV function being minimized, the BIC function is minimized. The BIC function is

$$f(\lambda) = -2 \times \text{log likelihood} + k \ln N$$

where k is the number of coefficients in model(λ) and N is the number of observations.

Just like selection using CV, selection using BIC searches for the minimum along a grid of λ's, starting with large λ's and moving toward smaller λ's. The λ grid is set up exactly the same way as it is for CV, and all the options to control the initialization of the grid that were described earlier work in exactly the same manner.

The criterion for identifying the minimum with BIC is similar to that for CV. The main difference is that a minimum λ^* will be identified when there are only two λ's on both sides of λ^* that have values of $f(\lambda)$ that are larger than $f(\lambda^*)$. CV requires three λ's for linear models and five for nonlinear models.

The stopping rules are the same for BIC as they are for CV, and the suboptions `stopok`, `strict`, `gridminok`, and `alllambdas` can be specified with `selection(bic)`, and all work the same way. To change the tolerance for identifying the minimum, you set `bictolerance()`, rather than `cvtolerance()`. See [LASSO] **lasso**.

Because the BIC function is computed analytically, there is no random component to its computation, unlike CV. This means that BIC is typically much faster than CV. However, this is not always true. The BIC function could have a flatter tail than the CV function and have to search more λ's in the grid. However, simulations seem to indicate that BIC typically yields a larger λ^* than CV and so typically selects fewer covariates than CV. In simulations, the number selected is typically close to but more than the number selected by plugin.

Also see

[LASSO] **lasso** — Lasso for prediction and model selection

[LASSO] **lasso examples** — Examples of lasso for prediction

[LASSO] **bicplot** — Plot Bayesian information criterion function after lasso

[LASSO] **cvplot** — Plot cross-validation function after lasso

[LASSO] **lassocoef** — Display coefficients after lasso estimation results

[LASSO] **lassogof** — Goodness of fit after lasso for prediction

[LASSO] **lassoknots** — Display knot table after lasso estimation

[LASSO] **lassoselect** — Select lambda after lasso

lassogof — Goodness of fit after lasso for prediction

Description

`lassogof` calculates goodness of fit of predictions after `lasso`, `sqrtlasso`, and `elasticnet`. It also calculates goodness of fit after `regress`, `logit`, `probit`, `poisson`, and `stcox` estimations for comparison purposes. For linear models, mean squared error of the prediction and R^2 are displayed. For logit, probit, Poisson, and Cox models, deviance and deviance ratio are shown.

Quick start

See goodness of fit for current lasso result using penalized coefficient estimates

```
    lassogof
```

See goodness of fit for current lasso result using postselection coefficient estimates

```
    lassogof, postselection
```

See goodness of fit for four stored estimation results

```
    lassogof mylasso mysqrtlasso myelasticnet myregress
```

See goodness of fit for all stored estimation results

```
    lassogof *
```

Randomly split sample into two, fit a lasso on the first sample, and calculate goodness of fit separately for both samples

```
    splitsample, generate(sample) nsplit(2)
    lasso linear y x* if sample == 1
    lassogof, over(sample)
```

Menu

Statistics > Postestimation

Syntax

lassogof [*namelist*] [*if*] [*in*] [, *options*]

namelist is a name of a stored estimation result, a list of names, `_all`, or `*`. `_all` and `*` mean the same thing. See [R] **estimates store**.

options	Description
Main	
`penalized`	use penalized (shrunken) coefficient estimates; the default
`postselection`	use postselection coefficient estimates
`over(`*varname*`)`	display goodness of fit for samples defined by *varname*
`noweights`	do not use weights when calculating goodness of fit

`collect` is allowed; see [U] **11.1.10 Prefix commands**.

Options

Main

`penalized` specifies that the penalized coefficient estimates be used to calculate goodness of fit. Penalized coefficients are those estimated by lasso with shrinkage. This is the default.

`postselection` specifies that the postselection coefficient estimates be used to calculate goodness of fit. Postselection coefficients are estimated by taking the covariates selected by lasso and reestimating the coefficients using an unpenalized estimator—namely, an ordinary linear regression, logistic regression, probit model, Poisson regression, or Cox regression as appropriate.

`over(`*varname*`)` specifies that goodness of fit be calculated separately for groups of observations defined by the distinct values of *varname*. Typically, this option would be used when the lasso is fit on one sample and one wishes to compare the fit in that sample with the fit in another sample.

`noweights` specifies that any weights used to estimate the lasso be ignored in the calculation of goodness of fit.

Remarks and examples

`lassogof` is intended for use on out-of-sample data. That is, on data different from the data used to fit the lasso.

There are two ways to do this. One is to randomly split your data into two subsamples before fitting a lasso model. The examples in this entry show how to do this using `splitsample`.

The other way is to load a different dataset in memory and run `lassogof` with the lasso results on it. The steps for doing this are as follows.

1. Load the data on which you are going to fit your model.

```
. use datafile1
```

2. Run `lasso` (or `sqrtlasso` or `elasticnet`).

```
. lasso ...
```

3. Save the results in a file.

```
. estimates save filename
```

4. Load the data for testing the prediction.

```
. use datafile2, clear
```

5. Load the saved results, making them the current (active) estimation results.

```
. estimates use filename
```

6. Run `lassogof`.

```
. lassogof
```

▷ Example 1: Comparing fit in linear models

We will show how to use `lassogof` after `lasso linear`.

Here is an example using `lasso` from [LASSO] **lasso examples**. We load the data and make the `vl` variable lists active.

```
. use https://www.stata-press.com/data/r19/fakesurvey_vl
(Fictitious survey data with vl)
. vl rebuild
Rebuilding vl macros ...
  (output omitted)
```

We now use `splitsample` to generate a variable indicating the two subsamples.

```
. set seed 1234
. splitsample, generate(sample) nsplit(2)
. label define svalues 1 "Training" 2 "Testing"
. label values sample svalues
```

We run lasso on the first subsample and set the random-number seed using the `rseed()` option so we can reproduce our results.

```
. lasso linear q104 ($idemographics) $ifactors $vlcontinuous
> if sample == 1, rseed(1234)
  (output omitted)
Lasso linear model                         No. of obs        =        458
                                           No. of covariates =        277
Selection: Cross-validation                No. of CV folds   =         10

------------------------------------------------------------------------------
             |                                No. of      Out-of-      CV mean
             |                               nonzero       sample   prediction
          ID |     Description      lambda     coef.    R-squared        error
-------------+----------------------------------------------------------------
           1 |    first lambda    .8978025         4       0.0147     16.93341
          18 |   lambda before    .1846342        42       0.2953     12.10991
        * 19 | selected lambda    .1682318        49       0.2968     12.08516
          20 |    lambda after    .1532866        55       0.2964     12.09189
          23 |     last lambda    .1159557        74       0.2913     12.17933
------------------------------------------------------------------------------
* lambda selected by cross-validation.
. estimates store linearcv
```

After the command finished, we used `estimates store` to store the results in memory so we can later compare these results with those from other lassos.

We are now going to run an adaptive lasso, which we do by specifying the option `selection(adaptive)`.

```
. lasso linear q104 ($idemographics) $ifactors $vlcontinuous
> if sample == 1, rseed(4321) selection(adaptive)
  (output omitted)
Lasso linear model                        No. of obs         =        458
                                          No. of covariates  =        277
Selection: Adaptive                       No. of lasso steps =          2
Final adaptive step results

──────────────────────────────────────────────────────────────────────────
        │                                No. of   Out-of-      CV mean
        │                               nonzero    sample   prediction
     ID │     Description      lambda     coef.  R-squared        error
────────┼─────────────────────────────────────────────────────────────────
     25 │    first lambda    48.55244         4     0.0101     17.01083
     77 │   lambda before    .3847698        46     0.3985     10.33691
   * 78 │ selected lambda    .3505879        46     0.3987     10.33306
     79 │    lambda after    .3194427        47     0.3985     10.33653
    124 │     last lambda    .0048552        59     0.3677     10.86697
──────────────────────────────────────────────────────────────────────────
* lambda selected by cross-validation in final adaptive step.
. estimates store linearadaptive
```

We want to see which performs better for out-of-sample prediction. We specify the `over()` option with the name of our sample indicator variable, `sample`. We specify the `postselection` option because for linear models, postselection coefficients are theoretically slightly better for prediction than the penalized coefficients (which `lassogof` uses by default). See the discussion in *predict* in [LASSO] **lasso postestimation**.

```
. lassogof linearcv linearadaptive, over(sample) postselection
Postselection coefficients

─────────────────────────────────────────────────────────────
Name                sample │        MSE    R-squared      Obs
───────────────────────────┼─────────────────────────────────
linearcv                   │
                  Training │   8.652771       0.5065      503
                   Testing │   14.58354       0.2658      493
───────────────────────────┼─────────────────────────────────
linearadaptive             │
                  Training │   8.637575       0.5057      504
                   Testing │   14.70756       0.2595      494
─────────────────────────────────────────────────────────────
```

The ordinary lasso did a little better in this case than adaptive lasso.

◁

▷ Example 2: Comparing fit in logit and probit models

We fit a logit model on the same data we used in the previous example.

```
. lasso logit q106 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234)
  (output omitted)
Lasso logit model                         No. of obs        =        458
                                          No. of covariates =        277
Selection: Cross-validation               No. of CV folds   =         10

--------------------------------------------------------------------------
         |                                No. of      Out-of-      CV mean
         |                               nonzero       sample
      ID |     Description      lambda     coef.   dev. ratio     deviance
---------+----------------------------------------------------------------
       1 |    first lambda    .1155342         0      -0.0004     1.384878
      22 |   lambda before    .0163767        65       0.1857     1.127315
    * 23 | selected lambda    .0149218        69       0.1871     1.125331
      24 |    lambda after    .0135962        73       0.1864     1.126333
      27 |     last lambda     .010285        88       0.1712     1.147343
--------------------------------------------------------------------------
* lambda selected by cross-validation.
. estimates store logit
```

Let's now fit a probit model.

```
. lasso probit q106 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234)
  (output omitted)
Lasso probit model                        No. of obs        =        458
                                          No. of covariates =        277
Selection: Cross-validation               No. of CV folds   =         10

--------------------------------------------------------------------------
         |                                No. of      Out-of-      CV mean
         |                               nonzero       sample
      ID |     Description      lambda     coef.   dev. ratio     deviance
---------+----------------------------------------------------------------
       1 |    first lambda    .1844415         0      -0.0004     1.384877
      21 |   lambda before    .0286931        61       0.1820     1.132461
    * 22 | selected lambda    .0261441        64       0.1846     1.128895
      23 |    lambda after    .0238215        70       0.1841     1.129499
      26 |     last lambda    .0180201        87       0.1677     1.152188
--------------------------------------------------------------------------
* lambda selected by cross-validation.
. estimates store probit
```

We look at how they did for out-of-sample prediction.

```
. lassogof logit probit, over(sample)
Penalized coefficients
-----------------------------------------------------------------------------
                            |                     Deviance
Name                 sample |       Deviance         ratio             Obs
-----------------------------------------------------------------------------
logit                       |
                   Training |       .8768969        0.3674             499
                    Testing |       1.268346        0.0844             502
-----------------------------------------------------------------------------
probit                      |
                   Training |       .8833892        0.3627             500
                    Testing |        1.27267        0.0812             503
-----------------------------------------------------------------------------
```

They both did not do very well. The out-of-sample deviance ratios were notably worse than the in-sample values. The deviance ratio for nonlinear models is analogous to R^2 for linear models. See *Methods and formulas* for the formal definition.

We did not specify the `postselection` option in this case because there are no theoretical grounds for using postselection coefficients for prediction with nonlinear models.

◁

Stored results

`lassogof` stores the following in `r()`:

Macros	
`r(names)`	names of estimation results displayed
`r(over_var)`	name of the `over()` variable
`r(over_levels)`	levels of the `over()` variable
Matrices	
`r(table)`	matrix containing the values displayed

Methods and formulas

`lassogof` reports the mean squared error (MSE) and the R^2 measures of fit for linear models. It reports the deviance and the deviance ratio for `logit`, `probit`, `poisson`, and `cox` models. The deviance ratio is also known as D^2 in the literature.

See Wooldridge (2020, 720) for more about MSE and Wooldridge (2020, 76–77) for more about R^2. The deviance measures are described in Hastie, Tibshirani, and Wainwright (2015, 29–33) and McCullagh and Nelder (1989, 33–34). For the `cox` model deviance, see Simon, Friedman, Hastie, and Tibshirani (2011).

In the formulas below, we use $\texttt{xb}_i$ to denote the linear prediction for the ith observation. By default, the lasso penalized coefficients $\widehat{\beta}$ are used to compute $\texttt{xb}_i$. Specifying the option `postselection` causes the postselection estimates $\widetilde{\widehat{\beta}}$ to be used to compute $\texttt{xb}_i$. See *predict* in [LASSO] **lasso postestimation** for a discussion of penalized estimates and postselection estimates.

We also use the following notation. y_i denotes the ith observation of the outcome. w_i is the weight applied to the ith observation; $w_i = 1$ if no weights were specified in the estimation command or if option `noweights` was specified in `lassogof`. N is the number of observations in the sample over which the goodness-of-fit statistics are computed. If frequency weights were specified at estimation $N_s = \sum_{i=1}^N w_i$; otherwise, $N_s = N$.

The formulas for the measures reported after linear models are

$$R^2 = 1 - \text{RSS}/\text{TSS}$$

$$\text{MSE} = 1/N_s \text{RSS}$$

where

$$\text{RSS} = \sum_{i=1}^N w_i (y_i - \texttt{xb}_i)^2$$

$$\text{TSS} = \sum_{i=1}^N w_i (y_i - \overline{y})^2$$

$$\overline{y} = \frac{1}{N_s} \sum_{i=1}^N w_i y_i$$

The deviance ratio D^2 is given by

$$D^2 = \frac{D_{\text{null}} - D}{D_{\text{null}}}$$

where D_{null} is the deviance calculated when only a constant term is included in the model and D is the deviance of the full model.

The formulas for the deviance and for D_{null} vary by model.

For `logit`, the deviance and the D_{null} are

$$D = -\frac{2}{N_s} \sum_{i=1}^N w_i \left[\tilde{y}_i \texttt{xb}_i + \ln\{1 + \exp(\texttt{xb}_i)\} \right]$$

$$D_{\text{null}} = -\frac{2}{N_s} \sum_{i=1}^N w_i \{ \tilde{y}_i \ln \overline{y} + (1 - \tilde{y}_i) \ln(1 - \overline{y}) \}$$

$$\tilde{y}_i = \begin{cases} 1 & y_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\overline{y} = \frac{1}{N_s} \sum_{i=1}^N w_i \tilde{y}_i$$

For probit, the deviance and the D_{null} are

$$D = -\frac{2}{N_s}\sum_{i=1}^{N} w_i\left[\tilde{y}_i \ln\{\Phi(\texttt{xb}_i)\} + (1-\tilde{y}_i)\ln\{1-\Phi(\texttt{xb}_i)\}\right]$$

$$D_{\text{null}} = -\frac{2}{N_s}\sum_{i=1}^{N} w_i\{\tilde{y}_i \ln\overline{y} + (1-\tilde{y}_i)\ln(1-\overline{y})\}$$

$$\tilde{y}_i = \begin{cases} 1 & y_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\overline{y} = \frac{1}{N_s}\sum_{i=1}^{N} s_i w_i \tilde{y}_i$$

For poisson, the deviance and the D_{null} are

$$D = -\frac{2}{N_s}\sum_{i=1}^{N} w_i\{y_i\texttt{xb}_i - \exp(\texttt{xb}_i) - v_i\}$$

$$v_i = \begin{cases} 0 & \text{if } y_i = 0 \\ y_i \ln y_i - y_i & \text{otherwise} \end{cases}$$

$$D_{\text{null}} = -\frac{2}{N_s}\sum_{i=1}^{N} w_i(y_i \ln\overline{y} - \overline{y} - v_i)$$

$$\overline{y} = \frac{1}{N_s}\sum_{i=1}^{N} w_i y_i$$

For cox, the deviance and the D_{null} are

$$\begin{aligned}
D &= 2\left(l_{\text{saturated}} - l\right)\\
D_{\text{null}} &= 2\left(l_{\text{saturated}} - l_{\text{null}}\right)\\
l_{\text{saturated}} &= -\frac{1}{N_s}\sum_{j=1}^{N_f} d_j \log\left(d_j\right)\\
l &= -\frac{1}{N_s}\sum_{j=1}^{N_f}\sum_{i\in D_j}\left[w_i(\texttt{xb}_i) - w_i \log\left\{\sum_{\ell\in R_j} w_\ell \exp(\texttt{xb}_\ell)\right\}\right]\\
l_{\text{null}} &= -\frac{1}{N_s}\sum_{j=1}^{N_f} d_j \log\left(\sum_{i\in R_j} w_i\right)\\
d_j &= \sum_{i\in D_j} w_i
\end{aligned}$$

where j indexes the ordered failure times $t_{(j)}$, $j = 1,\dots,N_f$; D_j is the set of observations that fail at $t_{(j)}$; R_j is the set of observations k that are at risk at time $t_{(j)}$ (that is, all k such that $t_{0k} < t_{(j)} \leq t_k$, and t_{0k} is the entry time for the kth observation).

References

Hastie, T. J., R. J. Tibshirani, and M. Wainwright. 2015. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Boca Raton, FL: CRC Press. https://doi.org/10.1201/b18401.

McCullagh, P., and J. A. Nelder. 1989. *Generalized Linear Models*. 2nd ed. London: Chapman and Hall/CRC.

Simon, N., J. H. Friedman, T. J. Hastie, and R. J. Tibshirani. 2011. Regularization paths for Cox's proportional hazards model via coordinate descent. *Journal of Statistical Software* 39: art. 5. https://doi.org/10.18637/jss.v039.i05.

Wooldridge, J. M. 2020. *Introductory Econometrics: A Modern Approach*. 7th ed. Boston: Cengage.

Also see

[LASSO] **lasso** — Lasso for prediction and model selection

[LASSO] **lassoknots** — Display knot table after lasso estimation

[LASSO] **lasso postestimation** — Postestimation tools for lasso for prediction

Postestimation commands

The following postestimation commands are of special interest after the `ds`, `po`, and `xpo` commands:

Command	Description
* `bicplot`	plot Bayesian information criterion function
* `coefpath`	plot path of coefficients
* `cvplot`	plot cross-validation function
`lassocoef`	display selected coefficients
`lassoinfo`	information about lasso estimation results
`lassoknots`	knot table of coefficient selection and measures of fit
* `lassoselect`	select alternative λ^* (and α^* for `elasticnet`)

*`bicplot` requires that the selection method of the lasso be `selection(bic)`. `cvplot` requires that the selection method of the lasso be `selection(cv)` or `selection(adaptive)`. `lassoselect` requires that the selection method of the lasso be `selection(bic)`, `selection(cv)`, or `selection(adaptive)`. See [LASSO] **lasso options**.

The following standard postestimation commands are also available:

Command	Description
`contrast`	contrasts and ANOVA-style joint tests of parameters
`estat summarize`	summary statistics for the estimation sample
`estat vce`	variance–covariance matrix of the estimators (VCE)
`estimates`	cataloging estimation results
`etable`	table of estimation results
`lincom`	point estimates, standard errors, testing, and inference for linear combinations of parameters
`nlcom`	point estimates, standard errors, testing, and inference for nonlinear combinations of parameters
`predict`	linear predictions
`predictnl`	point estimates for generalized predictions
`pwcompare`	pairwise comparisons of parameters
`test`	Wald tests of simple and composite linear hypotheses
`testnl`	Wald tests of nonlinear hypotheses

predict

Description for predict

predict creates a new variable containing the linear form $\mathbf{X}\widehat{\boldsymbol{\beta}}'$, where $\widehat{\boldsymbol{\beta}}$ is the vector of estimated coefficients of the variables of interest and does not include a constant term. This is the only type of prediction available after the ds, po, and xpo commands.

Menu for predict

Statistics > Postestimation

Syntax for predict

predict [*type*] *newvar* [*if*] [*in*]

Remarks and examples

After the ds, po, and xpo estimation commands, predict computes only the linear form $\mathbf{X}\widehat{\boldsymbol{\beta}}'$. So, for example, you need to type only

```
    . predict xbhat
```

The formulation of the lasso inferential models does not lend itself to making predictions for means, probabilities, or counts.

Also see

[LASSO] **Lasso inference intro** — Introduction to inferential lasso models

[LASSO] **Inference examples** — Examples and workflow for inference

[LASSO] **dslogit** — Double-selection lasso logistic regression

[LASSO] **dspoisson** — Double-selection lasso Poisson regression

[LASSO] **dsregress** — Double-selection lasso linear regression

[LASSO] **poivregress** — Partialing-out lasso instrumental-variables regression

[LASSO] **pologit** — Partialing-out lasso logistic regression

[LASSO] **popoisson** — Partialing-out lasso Poisson regression

[LASSO] **poregress** — Partialing-out lasso linear regression

[LASSO] **xpoivregress** — Cross-fit partialing-out lasso instrumental-variables regression

[LASSO] **xpologit** — Cross-fit partialing-out lasso logistic regression

[LASSO] **xpopoisson** — Cross-fit partialing-out lasso Poisson regression

[LASSO] **xporegress** — Cross-fit partialing-out lasso linear regression

[U] 20 Estimation and postestimation commands

lassoinfo — Display information about lasso estimation results

Description   Quick start   Menu   Syntax   Option
Remarks and examples   Stored results   Also see

Description

`lassoinfo` displays basic information about the lasso or lassos fit by all commands that fit lassos.

Quick start

After any command that fits lassos

```
lassoinfo
```

`dsregress` was run and the results stored under the name `mygreatmodel` using `estimates store`; show information about all the lassos in `mygreatmodel`

```
lassoinfo mygreatmodel
```

Same as above, but three models were stored

```
lassoinfo mygreatmodel mygoodmodel myfairmodel
```

After an `xpo` command, show information about every single lasso fit

```
lassoinfo, each
```

Menu

Statistics > Postestimation

Syntax

For all lasso estimation results

`lassoinfo` [*namelist*]

For `xpo` *estimation results*

`lassoinfo` [*namelist*] [`, each`]

namelist is a name of a stored estimation result, a list of names, `_all`, or `*`. `_all` and `*` mean the same thing. See [R] **estimates store**.

`collect` is allowed; see [U] **11.1.10 Prefix commands**.

Option

`each` applies to `xpo` models only. It specifies that information be shown for each lasso for each cross-fit fold to be displayed. If `resample` was specified, then information is shown for each lasso for each cross-fit fold in each resample. By default, summary statistics are shown for the lassos.

Remarks and examples

lassoinfo is intended for use after ds, po, xpo commands and after telasso to see basic information about the lassos they fit. It is a good idea to *always* run lassoinfo after these commands to see how many variables were selected in each lasso.

Running lassoinfo is a first step toward doing a sensitivity analysis. The lassos listed by lassoinfo can be examined using coefpath, cvplot, lassocoef, lassoknots, and lassoselect.

▷ Example 1: lasso

lassoinfo works after lasso, sqrtlasso, and elasticnet, but it does not display much useful information for these commands.

Here is an example using lasso from [LASSO] **lasso examples**. We load the data and make the vl variable lists active.

```
. use https://www.stata-press.com/data/r19/fakesurvey_vl
(Fictitious survey data with vl)
. vl rebuild
Rebuilding vl macros ...
  (output omitted )
```

We fit the lasso.

```
. lasso linear q104 $idemographics $ifactors $vlcontinuous, rseed(1234)
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .9090511   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 18.33331
  (output omitted )
Grid value 28:    lambda = .0737359   no. of nonzero coef. =  80
Folds: 1...5....10   CVF = 11.92887
... cross-validation complete ... minimum found
Lasso linear model                          No. of obs        =        914
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds   =         10

───────────────────────────────────────────────────────────────────────────
           │                                No. of      Out-of-      CV mean
           │                               nonzero       sample   prediction
        ID │     Description      lambda     coef.    R-squared        error
───────────┼───────────────────────────────────────────────────────────────
         1 │    first lambda    .9090511         0      -0.0010     18.33331
        23 │   lambda before    .1174085        58       0.3543     11.82553
      * 24 │ selected lambda    .1069782        64       0.3547     11.81814
        25 │    lambda after    .0974746        66       0.3545      11.8222
        28 │     last lambda    .0737359        80       0.3487     11.92887
───────────────────────────────────────────────────────────────────────────
* lambda selected by cross-validation.
```

`lassoinfo` tells us nothing new.

```
. lassoinfo

    Estimate: active
     Command: lasso

────────────────────────────────────────────────────────────────────────────
             │                                                     No. of
   Dependent │              Selection  Selection                  selected
    variable │       Model     method  criterion      lambda     variables
─────────────┼──────────────────────────────────────────────────────────────
        q104 │      linear         cv    CV min.    .1069782            64
────────────────────────────────────────────────────────────────────────────
```

Replaying the command gives more information.

```
. lasso

Lasso linear model                          No. of obs        =        914
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds   =         10

──────────────────────────────────────────────────────────────────────────────
         │                                No. of      Out-of-      CV mean
         │                               nonzero       sample   prediction
      ID │     Description      lambda     coef.    R-squared        error
─────────┼────────────────────────────────────────────────────────────────────
       1 │    first lambda    .9090511         0      -0.0010     18.33331
      23 │   lambda before    .1174085        58       0.3543     11.82553
    * 24 │ selected lambda    .1069782        64       0.3547     11.81814
      25 │    lambda after    .0974746        66       0.3545      11.8222
      28 │     last lambda    .0737359        80       0.3487     11.92887
──────────────────────────────────────────────────────────────────────────────
* lambda selected by cross-validation.
```

◁

▷ Example 2: dsregress

`lassoinfo` gives important information after the `ds`, `po`, and `xpo` commands.

We load the data used in [LASSO] **lasso examples**. See that entry for details about the data.

```
. use https://www.stata-press.com/data/r19/fakesurvey_vl, clear
(Fictitious survey data with vl)

. vl rebuild
Rebuilding vl macros ...
  (output omitted)
```

We are going to fit a `dsregress` model with `q104` as our dependent variable and variables of interest `q41` and `q22`. These variables of interest are currently in the variable lists `factors` and `vlcontinuous`, which we will use to specify the control variables. So we need to move them out of these variable lists.

```
. vl modify factors = factors - (q41)
note: 1 variable removed from $factors.

. vl move (q22) vlother
note: 1 variable specified and 1 variable moved.
  (output omitted)

. vl rebuild
Rebuilding vl macros ...
  (output omitted)
```

After we moved the variables out of the variable lists, we typed `vl rebuild` to update the variable list `ifactors` created from `factors`. See [D] **vl** for details.

We fit our `dsregress` model using cross-validation to select λ^*'s in the lassos.

```
. dsregress q104 i.q41 q22,
> controls(($idemographics) $ifactors $vlcontinuous)
> selection(cv) rseed(1234)
Estimating lasso for q104 using cv
Estimating lasso for 1bn.q41 using cv
Estimating lasso for q22 using cv
Double-selection linear model         Number of obs               =        914
                                      Number of controls          =        274
                                      Number of selected controls =        123
                                      Wald chi2(2)                =      10.96
                                      Prob > chi2                 =     0.0042

------------------------------------------------------------------------------
             |               Robust
        q104 | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         q41 |
        Yes  |   .6003918   .2848483     2.11   0.035     .0420994    1.158684
         q22 |  -.0681067   .0306219    -2.22   0.026    -.1281246   -.0080888
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

`lassoinfo` shows us how many variables were selected in each lasso.

```
. lassoinfo
    Estimate: active
     Command: dsregress

-----------------------------------------------------------------
             |                                            No. of
             |         Selection  Selection              selected
    Variable |   Model    method  criterion     lambda  variables
-------------+---------------------------------------------------
        q104 |  linear        cv    CV min.   .1116376         63
     1bn.q41 |  linear        cv    CV min.   .0135958         68
         q22 |  linear        cv    CV min.   .1624043         49
-----------------------------------------------------------------
```

lassoinfo also gives useful information after fitting the model using the default selection(plugin).

```
. dsregress q104 i.q41 q22, controls(($idemographics) $ifactors $vlcontinuous)

Estimating lasso for q104 using plugin
Estimating lasso for 1bn.q41 using plugin
Estimating lasso for q22 using plugin

Double-selection linear model         Number of obs               =        914
                                      Number of controls          =        274
                                      Number of selected controls =         33
                                      Wald chi2(2)                =      18.72
                                      Prob > chi2                 =     0.0001

------------------------------------------------------------------------------
             |               Robust
        q104 | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         q41 |
         Yes |   .8410538   .2691082     3.13   0.002     .3136114    1.368496
         q22 |  -.0878443   .0310435    -2.83   0.005    -.1486884   -.0270001
------------------------------------------------------------------------------

Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.

. lassoinfo

    Estimate: active
     Command: dsregress

------------------------------------------------------
            |                                   No. of
            |              Selection          selected
   Variable |      Model      method    lambda variables
------------+-----------------------------------------
       q104 |     linear      plugin  .1467287        18
    1bn.q41 |     linear      plugin  .1467287        16
        q22 |     linear      plugin  .1467287        15
------------------------------------------------------
```

See [LASSO] **lassoselect**, where we continue this example and do a sensitivity analysis to examine the differences between the lassos fit using cross-validation and the lassos fit using the plugin estimator.

◁

▷ Example 3: poivregress

We want to show you some differences that arise when you fit models containing endogenous variables using `poivregress` and `xpoivregress`.

We will not describe the data or the model here. See [LASSO] **Inference examples**.

We load the data,

```
. use https://www.stata-press.com/data/r19/mroz2, clear
```

set `vl` variable lists,

```
. vl create vars      = (kidslt6 kidsge6 age husage city exper)
note: $vars initialized with 6 variables.
. vl substitute vars2 = c.vars c.vars#c.vars
. vl create iv        = (huseduc motheduc fatheduc)
note: $iv initialized with 3 variables.
. vl substitute iv2   = c.iv c.iv#c.iv
```

and fit our model using `poivregress`.

```
. poivregress lwage (educ = $iv2), controls($vars2) selection(cv) rseed(12345)
Estimating lasso for lwage using cv
Estimating lasso for educ using cv
Estimating lasso for pred(educ) using cv
Partialing-out IV linear model      Number of obs                 =        428
                                    Number of controls            =         27
                                    Number of instruments         =          9
                                    Number of selected controls   =         16
                                    Number of selected instruments =         4
                                    Wald chi2(1)                  =      11.10
                                    Prob > chi2                   =     0.0009

------------------------------------------------------------------------------
             |               Robust
       lwage | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
        educ |   .0765154   .0229707     3.33   0.001     .0314936    .1215371
------------------------------------------------------------------------------
Endogenous: educ
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
. estimates store poivregresscv
```

We stored our estimation results using `estimates store`, and here we use `lassoinfo` with the name used to store them.

```
. lassoinfo poivregresscv

    Estimate: poivregresscv
     Command: poivregress

                                                           No. of
                          Selection  Selection            selected
     Variable      Model     method  criterion    lambda  variables

        lwage     linear         cv    CV min.  .0353704          3
         educ     linear         cv    CV min.  .0530428         10
   pred(educ)     linear         cv    CV min.   .013186         12
```

Note that we have two lassos for `educ` labeled by `lassoinfo` as `educ` and `pred(educ)`. `poivregress` and `xpoivregress` perform two lassos for each endogenous variable, one for the endogenous variable and one for its prediction. `lassoinfo` shows us how to refer to each of these lassos in other postestimation commands using the `for()` option. In this example, we would type `for(educ)` and `for(pred(educ))`, respectively.

◁

▷ Example 4: xporegress

The `xpo` commands fit many lassos. For each lasso fit by a `po` command, the corresponding `xpo` command fits `xfolds(#)` × `resample(#)` lassos. `lassoinfo` can be used to get information about these lassos.

We will not describe the data or the model here. See [LASSO] **Inference examples**.

We load the data,

```
. use https://www.stata-press.com/data/r19/breathe, clear
(Nitrogen dioxide and attention)
```

set `vl` variable lists,

```
. vl set
  (output omitted)
. vl move (siblings_old siblings_young) vlcontinuous
note: 2 variables specified and 2 variables moved.
  (output omitted)
. vl create mycontinuous   = vlcontinuous - (react no2_class)
note: $mycontinuous initialized with 10 variables.
. vl substitute mycontrols = i.vlcategorical mycontinuous
```

and fit our model using xporegress with the options xfolds(3) and resample(2).

```
. xporegress react no2_class, controls($mycontrols) xfolds(3) resample(2)
> selection(cv) rseed(12345)
Resample 1 of 2 ...
Cross-fit fold 1 of 3 ...
Estimating lassos: 1.
Resample 1 of 2 ...
Cross-fit fold 2 of 3 ...
Estimating lassos: 1.
Resample 1 of 2 ...
Cross-fit fold 3 of 3 ...
Estimating lassos: 1.
Resample 2 of 2 ...
Cross-fit fold 1 of 3 ...
Estimating lassos: 1.
Resample 2 of 2 ...
Cross-fit fold 2 of 3 ...
Estimating lassos: 1.
Resample 2 of 2 ...
Cross-fit fold 3 of 3 ...
Estimating lassos: 1.
Cross-fit partialing-out                 Number of obs               =      1,036
linear model                             Number of controls          =         32
                                         Number of selected controls =         27
                                         Number of folds in cross-fit =         3
                                         Number of resamples         =          2
                                         Wald chi2(1)                =      20.99
                                         Prob > chi2                 =     0.0000

------------------------------------------------------------------------------
             |               Robust
       react | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
   no2_class |   2.332193   .5090902     4.58   0.000     1.334394    3.329991
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

For each cross-fit fold and each resample, xporegress fits lassos. So it fit six lassos for the dependent variable, react, and six for the variable of interest, no2_class. lassoinfo summarizes the numbers of variables selected across these six lassos for react and no2_class.

```
. lassoinfo
    Estimate: active
     Command: xporegress

--------------------------------------------------------------
             |                           No. of selected variables
             |                Selection  -------------------------
    Variable |      Model        method      min   median      max
-------------+------------------------------------------------
   no2_class |     linear            cv       11       15       15
       react |     linear            cv        9       15       19
--------------------------------------------------------------
```

Specifying the option each gives us information on each lasso.

```
. lassoinfo, each

    Estimate: active
     Command: xporegress

                                                                          No. of
  Dependent                Selection Resample  xfold  Selection             sel.
   variable        Model      method   number    no.  criterion     lambda  var.

  no2_class       linear          cv        1      1    CV min.   .2663004    11
  no2_class       linear          cv        1      2    CV min.   .2860957    15
  no2_class       linear          cv        1      3    CV min.   .2887414    14
  no2_class       linear          cv        2      1    CV min.   .2337636    15
  no2_class       linear          cv        2      2    CV min.   .2824076    15
  no2_class       linear          cv        2      3    CV min.   .2515777    15
      react       linear          cv        1      1    CV min.    6.07542     9
      react       linear          cv        1      2    CV min.   1.704323    19
      react       linear          cv        1      3    CV min.   3.449884    15
      react       linear          cv        2      1    CV min.   6.034922     9
      react       linear          cv        2      2    CV min.    4.31785    16
      react       linear          cv        2      3    CV min.   4.096779    15
```

See [LASSO] **lassocoef** for an example where we list the variables selected by each lasso.

◁

Stored results

lassoinfo stores the following in r():

Macros

r(names)	names of estimation results displayed

Matrices

r(table)	matrix containing the numerical values displayed

Also see

[LASSO] **lassoselect** — Select lambda after lasso

[LASSO] **lasso postestimation** — Postestimation tools for lasso for prediction

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

lassoknots — Display knot table after lasso estimation

Description

`lassoknots` shows a table of knots after a lasso. Knots are the values of λ at which variables in the model change.

`lassoknots` displays the names of the variables added or removed as models are fit for successive λ's. When using cross-validation (CV) to select λ^*, `lassoknots` will display values of the CV function.

`lassoknots` also displays measures of fit. After viewing measures of fit, you can select an alternative λ^* using `lassoselect`.

When `telasso`, `ds`, `po`, and `xpo` commands fit models using `selection(cv)`, `selection(adaptive)`, or `selection(bic)` (see [LASSO] **lasso options**), `lassoknots` can be used to show the CV function (for `cv` and `adaptive`) or the BIC function or other measures of fit for each of the lassos computed.

`lassoknots` does work after `selection(plugin)` but only shows measures for the single λ^* estimated by the plugin formula.

Quick start

Show knot table after `lasso`, `sqrtlasso`, and `elasticnet`

```
lassoknots
```

Same as above, but show number of nonzero coefficients, out-of-sample R^2, and variables added or removed after a linear model

```
lassoknots, display(nonzero osr2 variables)
```

Same as above, but show in-sample R^2 and CV mean-prediction error in addition to out-of-sample R^2

```
lassoknots, display(osr2 r2 cvmpe)
```

After `lasso logit`, `lasso probit`, or `lasso poisson`, show out-of-sample mean-deviance ratio, in-sample deviance ratio, and Bayes information criterion (BIC)

```
lassoknots, display(cvdevratio devratio bic)
```

After a lasso fit with `selection(adaptive)`, show knot tables for all adaptive steps

```
lassoknots, steps
```

After a `ds` or `po` estimation with `selection(cv)` or `selection(adaptive)`, show the knot table for the lasso for the dependent variable `y`

```
lassoknots, for(y)
```

After `poivregress`, show the knot table for the lasso for the prediction of the endogenous variable `whatup`

```
lassoknots, for(pred(whatup))
```

After `xporegress` with option `resample`, show the knot table for the lasso for `x` for the 4th cross-fit fold of the 9th resample

```
lassoknots, for(x) xfold(4) resample(9)
```

After `telasso` estimation with `selection(cv)` or `selection(adaptive)`, show the knot table for the lasso for the outcome variable `y` at treatment level 1

```
lassoknots, for(y) tlevel(1)
```

Menu

Statistics > Postestimation

Syntax

After `lasso`, `sqrtlasso`, *and* `elasticnet`

`lassoknots` [, *options*]

After `ds` *and* `po`

`lassoknots, for(`*varspec*`)` [*options*]

After `xpo` *without* `resample`

`lassoknots, for(`*varspec*`) xfold(#)` [*options*]

After `xpo` *with* `resample`

`lassoknots, for(`*varspec*`) xfold(#) resample(#)` [*options*]

After `telasso` *for the outcome variable*

`lassoknots, for(`*varspec*`) tlevel(#)` [*options*]

After `telasso` *for the treatment variable*

`lassoknots, for(`*varspec*`)` [*options*]

After `telasso` *for the outcome variable with cross-fitting but without* `resample`

`lassoknots, for(`*varspec*`) tlevel(#) xfold(#)` [*options*]

After `telasso` *for the treatment variable with cross-fitting but without* `resample`

`lassoknots, for(`*varspec*`) xfold(#)` [*options*]

After `telasso` *for the outcome variable with cross-fitting and* `resample`

`lassoknots, for(`*varspec*`) tlevel(#) xfold(#) resample(#)` [*options*]

After `telasso` *for the treatment variable with cross-fitting and* `resample`

`lassoknots, for(`*varspec*`) xfold(#) resample(#)` [*options*]

varspec is *varname*, except after `poivregress` and `xpoivregress`, when it is either *varname* or `pred(`*varname*`)`.

options	Description
`display(`*di_opts*`)`	specify what to display; maximum of three *di_opts* options
`alllambdas`	show all λ's
`steps`	show all adaptive steps; `selection(adaptive)` only
`nolstretch`	do not stretch the width of the table to accommodate long variable names
* `for(`*varspec*`)`	lasso for *varspec*; `telasso`, `ds`, `po`, and `xpo` commands only
* `xfold(#)`	lasso for the #th cross-fit fold; `xpo` commands and `telasso` with `xfolds` only
* `resample(#)`	lasso for the #th resample; `xpo` commands and `telasso` with `resample` only
* `tlevel(#)`	lasso for the outcome model with the treatment level #; `telasso` only

*`for(`*varspec*`)` is required for all `ds`, `po`, and `xpo` commands and for `telasso`.
`xfold(#)` is required for all `xpo` commands and for `telasso` when the option `xfolds(#)` was specified.
`resample(#)` is required for `xpo` and for `telasso` when the option `resample(#)` was specified.
`tlevel(#)` is required for the outcome model in `telasso`.
`collect` is allowed; see **[U] 11.1.10 Prefix commands**.

di_opts	Description
`nonzero`	number of nonzero coefficients
`variables`	names of variables added or removed
`cvmd`	CV mean deviance (the CV function)
`cvdevratio`	CV mean-deviance ratio
`devratio`	in-sample deviance ratio
`bic`	BIC
`l1`	relative ℓ_1-norm of coefficients
`l2`	relative ℓ_2-norm squared of coefficients
Linear models only	
`cvmpe`	CV mean-prediction error (the CV function)
`osr2`	out-of-sample R^2
`r2`	in-sample R^2

Options

`display(`*di_opts*`)` specifies what to display in the knot table. A maximum of three *di_opts* options can be specified. For lassos fit using `selection(cv)` or `selection(adaptive)`, the default is `display(nonzero cvmpe variables)` for linear models and `display(nonzero cvmd variables)` for logit, probit, Poisson, and Cox models. For lassos fit using `selection(plugin)` or `selection(bic)`, the default is `display(nonzero r2 variables)` for linear models and `display(nonzero devratio variables)` for logit, probit, Poisson, and Cox models. The full set of *di_opts* is the following.

`nonzero` specifies that the number of nonzero coefficients be shown.

`variables` specifies that the names of variables added or removed at each knot be shown.

`cvmd` specifies that the CV mean deviance be shown. These are the values of the CV function that are searched for a minimum. For linear models, it is the same as the CV mean-prediction error given by `cvmpe`. `cvmd` is available only for lassos fit using `selection(cv)` or `selection(adaptive)`.

`cvdevratio` specifies that the CV mean-deviance ratio be shown. The CV mean-deviance ratio is an estimate of out-of-sample goodness of fit. As a measure of prediction performance, it is superior to `devratio`, the in-sample deviance ratio. It is typically between 0 and 1, but in some cases, it may be outside this range. For linear models, it is the same as out-of-sample R^2 given by `osr2`. `cvdevratio` is available only for lassos fit using `selection(cv)` or `selection(adaptive)`.

`devratio` specifies that the in-sample deviance ratio be shown. The in-sample deviance ratio is an indicator of in-sample goodness of fit. The in-sample deviance generalizes the in-sample R^2 to nonlinear models. As a measure of prediction performance, it is inferior to `cvdevratio`, the CV mean-deviance ratio. The in-sample deviance ratio is a poor measure of prediction performance because it does not capture the cost of including additional covariates for prediction. It is always between 0 and 1. For linear models, it is the same as in-sample R^2 given by `r2`.

`bic` specifies that the BIC be shown. Note that the BIC can be displayed for lassos fit using `selection(cv)` and `selection(adaptive)`, but the CV measures—`cvmd`, `cvdevratio`, and `cvmpe`—are not available for lassos fit using `selection(bic)`.

`l1` specifies that the relative ℓ_1-norm of coefficients be shown.

`l2` specifies that relative ℓ_2-norm squared of coefficients be shown.

Linear models only

`cvmpe` specifies that the CV mean-prediction error be shown. These are the values of the CV function that are searched for a minimum. `cvmpe` is available only for lassos fit using `selection(cv)` or `selection(adaptive)`.

`osr2` specifies that the out-of-sample R^2 be shown. The out-of-sample R^2 is an estimate of out-of-sample goodness of fit. As a measure of prediction performance, it is superior to `r2`, the in-sample R^2. It is typically between 0 and 1, but in some cases, it may be outside this range.

`r2` specifies that the in-sample deviance ratio be shown. The in-sample deviance ratio is an indicator of in-sample goodness of fit. As a measure of prediction performance, it is inferior to `osr2`, the out-of-sample R^2. The in-sample R^2 is a poor measure of prediction performance because it does not capture the cost of including additional covariates for prediction. It is always between 0 and 1.

`alllambdas` specifies that all λ's are to be shown, not just the knots. Measures at λ's that are not knots change slightly because the coefficient estimates change slightly. λ's that are not knots can be selected as λ^* by `lassoselect`; however, this is typically not done.

`steps` applies to `selection(adaptive)` only. When specified, λ's for all adaptive steps are shown. By default, λ's for only the last adaptive step are shown.

`nolstretch` specifies that the width of the table not be automatically widened to accommodate long variable names. When `nolstretch` is specified, names are abbreviated to make the table width no more than 79 characters. The default, `lstretch`, is to automatically widen the table up to the width of the Results window. To change the default, use `set lstretch off`.

`for(`*varspec*`)` specifies a particular lasso after `telasso` or after a `ds`, `po`, or `xpo` estimation command fit using the option `selection(cv)`, `selection(adaptive)`, or `selection(bic)`. For all commands except `poivregress` and `xpoivregress`, *varspec* is always *varname*.

For the `ds`, `po`, and `xpo` commands except `poivregress` and `xpoivregress`, *varspec* is either *depvar*, the dependent variable, or one of *varsofinterest* for which inference is done.

For `poivregress` and `xpoivregress`, *varspec* is either *varname* or `pred(`*varname*`)`. The lasso for *depvar* is specified with its *varname*. Each of the endogenous variables have two lassos, specified by *varname* and `pred(`*varname*`)`. The exogenous variables of interest each have only one lasso, and it is specified by `pred(`*varname*`)`.

For `telasso`, *varspec* is either the outcome variable or the treatment variable.

This option is required after `telasso` and after the `ds`, `po`, and `xpo` commands.

`xfold(`#`)` specifies a particular lasso after an `xpo` estimation command or after `telasso` when the option `xfolds(`#`)` was specified. For each variable to be fit with a lasso, K lassos are done, one for each cross-fit fold, where K is the number of folds. This option specifies which fold, where $\# = 1, 2, \dots, K$. `xfold(`#`)` is required after an `xpo` command and after `telasso` when the option `xfolds(`#`)` was specified.

`resample(`#`)` specifies a particular lasso after an `xpo` estimation command or after `telasso` fit using the option `resample(`#`)`. For each variable to be fit with a lasso, $R \times K$ lassos are done, where R is the number of resamples and K is the number of cross-fitting folds. This option specifies which resample, where $\# = 1, 2, \dots, R$. `resample(`#`)`, along with `xfold(`#`)`, is required after an `xpo` command and after `telasso` with resampling.

`tlevel(`#`)` specifies the lasso for the outcome variable at the specified treatment level after `telasso`. This option is required to refer to the outcome model after `telasso`.

Remarks and examples

Remarks are presented under the following headings:

Introduction

When a lasso is fit over a grid of λ's, it starts with the smallest λ that produces a model with no selected variables. This initial λ is the largest λ in the grid. Lasso steps to the next λ and fits a model for it. One or more variables are selected for this second λ (if no variables were selected, it would be the starting λ). Lasso steps to the third λ, and more variables may be selected, or the model may have the same variables as the model for the second λ.

In this way, lasso iterates across the grid of λ values. λ's at which the selected variables change are called "knots". Variables are not only added at a knot but also sometimes removed. Typically, when a variable is removed, one or more variables are added in its place. Usually, the number of nonzero coefficients increases monotonically as λ gets smaller but not always. Occasionally, the net number of variables in the model goes down, rather than up, in an iteration to a smaller λ.

`lassoknots` displays a table of the knots, showing the names of variables that enter and leave the models. The option `alllambdas` can be specified to display all the λ's in the grid. To view all variables selected at a particular λ, you can use `lassoselect` to select that λ and then `lassocoef` to list the variables and, optionally, the coefficients.

Selection methods `selection(cv)`, `selection(adaptive)`, `selection(bic)`, and `selection(none)` fit models for each λ in the grid. The method `selection(plugin)` calculates λ^* using a formula so there is only one λ.

Measures of fit

`lassoknots` will also display other measures. The methods `selection(cv)` and `selection(adaptive)` use CV. When CV is performed, `lassoknots` by default displays the number of nonzero coefficients, the CV function, and the names of variables that enter or leave the model.

Optionally, there are five other measures that can be displayed. For linear models, they are in-sample R^2 (`r2`), estimates of out-of-sample R^2 (`osr2`), the BIC (`bic`), relative ℓ_1-norm of coefficients (`l1`), and relative ℓ_2-norm squared of coefficients (`l2`).

For nonlinear models, in place of the R^2 measures, there are the analogous measures, the in-sample deviance ratio (`devratio`) and estimates of out-of-sample deviance ratio (`cvdevratio`).

The in-sample measures, BIC, and relative norms are available regardless of whether CV was done.

The out-of-sample R^2 and out-of-sample deviance ratio are not computed on out-of-sample data, but rather they are estimates of what these measures would be on out-of-sample data. The CV procedure provides these estimates.

In-sample measures versus estimates of out-of-sample measures

Estimates of out-of-sample measures are superior to in-sample measures.

Consider a linear lasso. The set of covariates that produces the smallest out-of-sample MSE is the set that produces the best predictions. CV is used to estimate out-of-sample MSE and select the set that produces the smallest estimate.

In contrast, we should not use in-sample MSE to select the set of covariates. In-sample MSE systematically underestimates out-of-sample prediction error. In-sample MSE can be made smaller and smaller simply by including more covariates (as long as they are not collinear with covariates already in the model). In-sample MSE does not capture the cost of including more covariates.

For the same reason, estimates of out-of-sample R^2 are superior to in-sample R^2 for linear models. For logit, probit, and Poisson models, estimates of out-of-sample deviance ratios are superior to in-sample deviance ratios.

See Hastie, Tibshirani, and Friedman (2009, sec. 7.2) for an introduction to a comparison of in-sample and out-of-sample measures of the predictive ability of a model.

BIC

Information criteria, like the BIC, have a term that penalizes for each additional parameter. Selecting the set of covariates that minimizes the BIC is another way to select a set of covariates that will predict well out of sample. Zhang, Li, and Tsai (2010) show that the λ selected by minimizing the BIC will select a set of covariates close to the true set under the conditions described in their article.

In practice, the BIC is more informative than the in-sample measures reported by `lassoknots` for `selection(plugin)` and `selection(none)`.

Examples

▷ Example 1: lasso linear

Here is an example using `lasso` from [LASSO] **lasso examples**. We load the data and make the `vl` variable lists active.

```
. use https://www.stata-press.com/data/r19/fakesurvey_vl
(Fictitious survey data with vl)

. vl rebuild
Rebuilding vl macros ...
  (output omitted)
```

We fit a lasso linear model.

```
. lasso linear q104 $idemographics $ifactors $vlcontinuous, rseed(1234)

10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .9090511   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 18.33331
  (output omitted)
Grid value 28:    lambda = .0737359   no. of nonzero coef. =  80
Folds: 1...5....10   CVF = 11.92887
... cross-validation complete ... minimum found

Lasso linear model                          No. of obs         =        914
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds    =         10

------------------------------------------------------------------------------
                  |                           No. of      Out-of-     CV mean
                  |                          nonzero       sample  prediction
       ID |     Description      lambda        coef.    R-squared       error
----------+-------------------------------------------------------------------
        1 |    first lambda    .9090511            0      -0.0010    18.33331
       23 |   lambda before    .1174085           58       0.3543    11.82553
     * 24 | selected lambda    .1069782           64       0.3547    11.81814
       25 |    lambda after    .0974746           66       0.3545     11.8222
       28 |     last lambda    .0737359           80       0.3487    11.92887
------------------------------------------------------------------------------
* lambda selected by cross-validation.
```

We run lassoknots.

```
. lassoknots

    ─────────────────────────────────────────────────────────────────────────────
             |                      No. of   CV mean |
             |                     nonzero     pred. |   Variables (A)dded, (R)emoved,
          ID |     lambda           coef.     error  |        or left (U)nchanged
    ─────────────────────────────────────────────────────────────────────────────
           2 |   .8282935               2  18.24362  |  A 0.q19      0.q88
           3 |   .7547102               4  17.99053  |  A 0.q85      3.q156
           5 |   .6265736               7  17.26211  |  A 0.q48      0.q73      0.q101
           6 |   .5709106              11   16.7744  |  A 4.q38      q31        q76
             |                                       |    q139
           7 |   .5201924              15  16.19275  |  A 0.q5       2.q34      0.q43
             |                                       |    0.q50
           8 |     .47398              16  15.58941  |  A q22
          11 |   .3585485              19  14.07708  |  A 0.q41      0.q56      2.q84
          12 |    .326696              22  13.69483  |  A 3.q16      0.q89      0.q118
          13 |   .2976732              25   13.3281  |  A 0.q91      age        0.gender
          14 |   .2712288              26  12.99274  |  A 3.q38
          16 |   .2251789              32  12.48904  |  A 0.q3       0.q49      0.q150
             |                                       |    2.q155     0.q160     q111
          18 |   .1869475              34  12.15245  |  A 2.q6       3.q78
          19 |   .1703396              39  12.03358  |  A 0.q14      0.q33      0.q126
             |                                       |    0.q147     0.q149
          20 |   .1552071              42  11.94361  |  A 0.q25      0.q82      1.q110
          21 |   .1414189              46  11.88652  |  A 0.q96      q20        3.q110
             |                                       |    1.q134
          22 |   .1288556              50  11.84693  |  A 0.q32      0.q102     1.q105
             |                                       |    0.q122
          23 |   .1174085              58  11.82553  |  A 0.q4       0.q7       1.q34
             |                                       |    0.q40      3.q84      q53
             |                                       |    q93        2.q134
        * 24 |   .1069782              64  11.81814  |  A 0.q51      0.q55      0.q75
             |                                       |    0.q77      q63        0.q115
          25 |   .0974746              66   11.8222  |  A 3.q6       0.q117
          26 |   .0888152              70  11.84669  |  A 0.q59      3.q95      q21
             |                                       |    0.q125
          27 |   .0809251              72  11.88463  |  A 0.q100     4.q155
          28 |   .0737359              80  11.92887  |  A 0.q13      0.q30      0.q68
             |                                       |    q52        q70        2.q110
             |                                       |    0.q153     0.q159
    ─────────────────────────────────────────────────────────────────────────────
    * lambda selected by cross-validation.
```

The table ends at the 28th λ. The default grid had 100 λ's. The iteration over the λ grid ended after a minimum of the CV function was found. There are other cases in which the iteration ends before the end of the grid is reached. See *The CV function* in [LASSO] **lasso** and [LASSO] **lasso fitting** for details.

The option `alllambdas` shows all the λ's for which models were fit. In this case, the first 28 λ's in the grid.

```
. lassoknots, alllambdas

                                   No. of    CV mean
                                  nonzero      pred.      Variables (A)dded, (R)emoved,
       ID       lambda              coef.      error           or left (U)nchanged

        1     .9090511                 0   18.33331     U
        2     .8282935                 2   18.24362     A 0.q19     0.q88
        3     .7547102                 4   17.99053     A 0.q85     3.q156
        4     .6876638                 4    17.6434     U
        5     .6265736                 7   17.26211     A 0.q48     0.q73      0.q101
        6     .5709106                11    16.7744     A 4.q38     q31        q76
                                                          q139
        7     .5201924                15   16.19275     A 0.q5      2.q34      0.q43
                                                          0.q50
        8       .47398                16   15.58941     A q22
        9     .4318729                16   15.01285     U
       10     .3935065                16   14.50648     U
       11     .3585485                19   14.07708     A 0.q41     0.q56      2.q84
       12      .326696                22   13.69483     A 3.q16     0.q89      0.q118
       13     .2976732                25    13.3281     A 0.q91     age        0.gender
       14     .2712288                26   12.99274     A 3.q38
       15     .2471336                26   12.71385     U
       16     .2251789                32   12.48904     A 0.q3      0.q49      0.q150
                                                          2.q155    0.q160     q111
       17     .2051746                32   12.30196     U
       18     .1869475                34   12.15245     A 2.q6      3.q78
       19     .1703396                39   12.03358     A 0.q14     0.q33      0.q126
                                                          0.q147    0.q149
       20     .1552071                42   11.94361     A 0.q25     0.q82      1.q110
       21     .1414189                46   11.88652     A 0.q96     q20        3.q110
                                                          1.q134
       22     .1288556                50   11.84693     A 0.q32     0.q102     1.q105
                                                          0.q122
       23     .1174085                58   11.82553     A 0.q4      0.q7       1.q34
                                                          0.q40     3.q84      q53
                                                          q93       2.q134
     * 24     .1069782                64   11.81814     A 0.q51     0.q55      0.q75
                                                          0.q77     q63        0.q115
       25     .0974746                66    11.8222     A 3.q6      0.q117
       26     .0888152                70   11.84669     A 0.q59     3.q95      q21
                                                          0.q125
       27     .0809251                72   11.88463     A 0.q100    4.q155
       28     .0737359                80   11.92887     A 0.q13     0.q30      0.q68
                                                          q52       q70        2.q110
                                                          0.q153    0.q159

  * lambda selected by cross-validation.
```

The λ's that are not knots have a `U` for unchanged in the variables column. At these λ's, the variables in the model do not change, but their coefficient estimates do. In this example, the selected λ^* is a knot, but frequently the selected λ^* will not be a knot.

We display the number of nonzero coefficients again, but this time with estimates of out-of-sample R^2 and in-sample R^2.

```
. lassoknots, display(nonzero osr2 r2)

─────────────────────────────────────────────────────────────────
         │                     No. of      Out-of-
         │                    nonzero       sample   In-sample
      ID │     lambda           coef.    R-squared   R-squared
─────────┼───────────────────────────────────────────────────────
         │
       2 │   .8282935               2       0.0039      0.0102
       3 │   .7547102               4       0.0177      0.0278
       5 │   .6265736               7       0.0575      0.0707
       6 │   .5709106              11       0.0841      0.1051
       7 │   .5201924              15       0.1159      0.1414
       8 │     .47398              16       0.1488      0.1790
      11 │   .3585485              19       0.2314      0.2635
      12 │    .326696              22       0.2523      0.2861
      13 │   .2976732              25       0.2723      0.3090
      14 │   .2712288              26       0.2906      0.3288
      16 │   .2251789              32       0.3181      0.3610
      18 │   .1869475              34       0.3365      0.3870
      19 │   .1703396              39       0.3430      0.3981
      20 │   .1552071              42       0.3479      0.4081
      21 │   .1414189              46       0.3510      0.4176
      22 │   .1288556              50       0.3532      0.4263
      23 │   .1174085              58       0.3543      0.4342
    * 24 │   .1069782              64       0.3547      0.4418
      25 │   .0974746              66       0.3545      0.4486
      26 │   .0888152              70       0.3532      0.4546
      27 │   .0809251              72       0.3511      0.4598
      28 │   .0737359              80       0.3487      0.4647
─────────────────────────────────────────────────────────────────
* lambda selected by cross-validation.
```

In-sample R^2 is significantly larger than the estimates of out-of-sample R^2. As we discussed in *In-sample measures versus estimates of out-of-sample measures* above, in-sample R^2 should not be used for assessing fit. It is, however, occasionally useful for exposing problems with the specification of the set of potential covariates. For example, suppose our dependent variable is log-income and we accidentally include income as a potential covariate. It will no doubt be selected, and we will see an R^2 of 1 or close to it. Seeing that, we realize we made a mistake in the specification of potential variables.

We run `lassoknots` again to display BIC and the relative norms of the coefficient vectors.

```
. lassoknots, display(l1 l2 bic)

                                                 
              │                                  Relative    Relative
              │                                        L1          L2
           ID │     lambda          BIC        length      length
       ───────┼──────────────────────────────────────────────────
            2 │   .8282935     5262.546        0.0084      0.0013
            3 │   .7547102      5259.79        0.0244      0.0060
            5 │   .6265736     5238.991        0.0696      0.0313
            6 │   .5709106     5231.834        0.1066      0.0544
            7 │   .5201924     5221.257        0.1449      0.0840
            8 │     .47398     5187.164        0.1903      0.1195
           11 │   .3585485     5108.273        0.3092      0.2504
           12 │    .326696     5100.274        0.3492      0.2982
           13 │   .2976732      5090.95        0.3948      0.3487
           14 │   .2712288     5071.186        0.4375      0.4001
           16 │   .2251789     5067.137        0.5179      0.4999
           18 │   .1869475     5042.754        0.5959      0.5949
           19 │   .1703396     5060.244        0.6344      0.6398
           20 │   .1552071     5065.277        0.6734      0.6834
           21 │   .1414189     5077.835        0.7133      0.7259
           22 │   .1288556     5091.401        0.7543      0.7677
           23 │   .1174085     5133.245        0.7955      0.8091
         * 24 │   .1069782     5161.662        0.8388      0.8503
           25 │   .0974746     5164.198        0.8805      0.8904
           26 │   .0888152     5181.477        0.9213      0.9286
           27 │   .0809251      5186.25        0.9606      0.9651
           28 │   .0737359     5232.569        1.0000      1.0000
       ──────────────────────────────────────────────────────────

  * lambda selected by cross-validation.
```

The relative norms are relative to the coefficient vector for the last λ. If we were using BIC to select λ^*, we would have chosen λ at ID $= 18$.

◁

▷ Example 2: lasso logit

We fit a lasso logit model using the same data as in the previous example.

```
. lasso logit q106 $idemographics $ifactors $vlcontinuous, rseed(1234)
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .0886291   no. of nonzero coef. =   0
Folds: 1...5....10   CVF = 1.386903
  (output omitted)
Grid value 27:    lambda = .0078899   no. of nonzero coef. =  87
Folds: 1...5....10   CVF = 1.103886
... cross-validation complete ... minimum found
Lasso logit model                           No. of obs         =        914
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds    =         10

-------------------------------------------------------------------------
         |                                No. of      Out-of-
         |                               nonzero       sample     CV mean
      ID |     Description      lambda     coef.   dev. ratio    deviance
---------+---------------------------------------------------------------
       1 |    first lambda    .0886291         0      -0.0004    1.386903
      23 |   lambda before    .0114469        68       0.2102    1.094953
    * 24 | selected lambda      .01043        76       0.2103     1.09471
      25 |    lambda after    .0095034        79       0.2091    1.096417
      27 |     last lambda    .0078899        87       0.2037    1.103886
-------------------------------------------------------------------------
* lambda selected by cross-validation.
```

The default `lassoknots` gives a table that is the same as that for a linear model, except that instead of CV mean-prediction error, CV mean deviance is shown. The CV function for logit (and probit and Poisson) is the CV mean deviance.

```
. lassoknots

------------------------------------------------------------------------------
       |                 No. of                |
       |                nonzero     CV mean    |   Variables (A)dded, (R)emoved,
    ID |     lambda       coef.    deviance    |         or left (U)nchanged
-------+---------------------------------------+------------------------------
     2 |   .0807555          3     1.38295    | A 0.q90     2.q134    0.q142
     3 |   .0735814          5     1.37237    | A 0.q8      q53
     4 |   .0670447          8    1.357427    | A 0.q68     0.q77     q22
     5 |   .0610886          9     1.33969    | A 0.q46
     6 |   .0556616         12    1.319525    | A 0.q13     2.q16     2.q95
     7 |   .0507168         14    1.299571    | A 1.q84     q20
     8 |   .0462113         18    1.279802    | A 0.q29     0.q133    0.q140
       |                                       |   1.q144
  (output omitted)
    23 |   .0114469         68    1.094953    | A 0.q26     0.q73     0.q118
  * 24 |     .01043         76     1.09471    | A 0.q4      q1        0.q50
       |                                       |   2.q65     3.q65     0.q83
       |                                       |   q24       1.q155
    25 |   .0095034         79    1.096417    | A q76       0.q108    0.q122
    26 |   .0086591         83     1.09945    | A 2.q6      0.q64     0.q100
       |                                       |   q132
    27 |   .0078899         87    1.103886    | A 0.q58     0.q74     0.q113
       |                                       |   q103
------------------------------------------------------------------------------
* lambda selected by cross-validation.
```

We can look at in-sample CV deviance ratio and estimates of out-of-sample CV deviance ratio. These are analogous to the linear in-sample R^2 and out-of-sample R^2. The in-sample CV deviance ratio is always between 0 and 1. The estimates of out-of-sample CV deviance ratio are usually, but not always, between 0 and 1.

```
. lassoknots, display(cvdevratio devratio bic)

───────────────────────────────────────────────────
        │                Out-of-  In-sample
        │                 sample   deviance
     ID │     lambda  dev. ratio      ratio       BIC
───────────────────────────────────────────────────
      2 │   .0807555      0.0024     0.0057  1287.176
      3 │   .0735814      0.0100     0.0180  1285.111
      4 │   .0670447      0.0208     0.0323  1287.477
      5 │   .0610886      0.0336     0.0488  1273.364
      6 │   .0556616      0.0482     0.0657  1272.417
      7 │   .0507168      0.0626     0.0835    1263.5
      8 │   .0462113      0.0768     0.1022  1267.165
  (output omitted)
     23 │   .0114469      0.2102     0.3209  1330.907
   * 24 │     .01043      0.2103     0.3297   1374.27
     25 │   .0095034      0.2091     0.3379  1384.306
     26 │   .0086591      0.2069     0.3461  1401.188
     27 │   .0078899      0.2037     0.3535  1419.149
───────────────────────────────────────────────────
* lambda selected by cross-validation.
```

◁

▷ Example 3: dsregress

We load the data used in [LASSO] **lasso examples**. See that entry for details about the data.

```
. use https://www.stata-press.com/data/r19/fakesurvey_vl, clear
(Fictitious survey data with vl)
. vl rebuild
Rebuilding vl macros ...
  (output omitted)
```

We are going to fit a `dsregress` model with `q104` as our dependent variable and variables of interest `q41` and `q22`. These variables of interest are currently in the variable lists `factors` and `vlcontinuous`, which we will use to specify the control variables. So we need to move them out of these variable lists.

```
. vl modify factors = factors - (q41)
note: 1 variable removed from $factors.
. vl move (q22) vlother
note: 1 variable specified and 1 variable moved.
  (output omitted)
. vl rebuild
Rebuilding vl macros ...
  (output omitted)
```

After we moved the variables out of the variable lists, we typed `vl rebuild` to update the variable list `ifactors` created from `factors`. See [D] **vl** for details.

We fit our dsregress model using the default plugin selection method.

```
. dsregress q104 i.q41 q22, controls(($idemographics) $ifactors $vlcontinuous)
Estimating lasso for q104 using plugin
Estimating lasso for 1bn.q41 using plugin
Estimating lasso for q22 using plugin
Double-selection linear model         Number of obs               =        914
                                      Number of controls          =        274
                                      Number of selected controls =         33
                                      Wald chi2(2)                =      18.72
                                      Prob > chi2                 =     0.0001

------------------------------------------------------------------------------
             |               Robust
        q104 | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         q41 |
         Yes |   .8410538   .2691082     3.13   0.002     .3136114    1.368496
         q22 |  -.0878443   .0310435    -2.83   0.005    -.1486884   -.0270001
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

lassoinfo shows the lassos that dsregress fit.

```
. lassoinfo
    Estimate: active
     Command: dsregress

-----------------------------------------------------------------
             |                                           No. of
             |                 Selection               selected
    Variable |       Model        method     lambda   variables
-------------+---------------------------------------------------
        q104 |      linear        plugin   .1467287          18
     1bn.q41 |      linear        plugin   .1467287          16
         q22 |      linear        plugin   .1467287          15
-----------------------------------------------------------------
```

The knot table for the lasso for the dependent variable q104 can be seen using the for(q104) option. We also show BIC and in-sample R^2.

```
. lassoknots, display(nonzero r2 bic) for(q104)

-------------------------------------------------------------
       |                     No. of
       |                    nonzero    In-sample
    ID |      lambda          coef.    R-squared          BIC
-------+-----------------------------------------------------
   * 1 |    .1467287             14       0.1623     5191.862
-------------------------------------------------------------
* lambda selected by plugin assuming heteroskedastic errors.
```

A lasso fit with plugin fits only one model for one λ. So that is all we get from lassoknots.

If we wanted to see the same table for the variable of interest i.q41, we would type

```
. lassoknots, display(nonzero r2 bic) for(1bn.q41)
```

In the for() option, we specify the variable name for the lasso exactly as it is shown in lassoinfo.

We run dsregress again, this time specifying selection(cv).

```
. dsregress q104 i.q41 q22,
> controls(($idemographics) $ifactors $vlcontinuous)
> selection(cv) rseed(1234)

Estimating lasso for q104 using cv
Estimating lasso for 1bn.q41 using cv
Estimating lasso for q22 using cv

Double-selection linear model         Number of obs               =        914
                                      Number of controls          =        274
                                      Number of selected controls =        123
                                      Wald chi2(2)                =      10.96
                                      Prob > chi2                 =     0.0042

─────────────┬────────────────────────────────────────────────────────────────
             │               Robust
        q104 │ Coefficient  std. err.      z    P>|z|     [95% conf. interval]
─────────────┼────────────────────────────────────────────────────────────────
             │
         q41 │
        Yes  │   .6003918   .2848483     2.11   0.035     .0420994    1.158684
         q22 │  -.0681067   .0306219    -2.22   0.026    -.1281246   -.0080888
─────────────┴────────────────────────────────────────────────────────────────

Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

lassoknots now shows knots up to the minimum and slightly passed it.

```
. lassoknots, display(nonzero cvmpe osr2) for(q104)

───────┬──────────────────────────────────────────────
       │                  No. of    CV mean    Out-of-
       │                 nonzero      pred.     sample
    ID │    lambda         coef.      error  R-squared
───────┼──────────────────────────────────────────────
     1 │   .864369             4    17.9727     0.0187
     2 │  .7875809             6   17.88282     0.0236
     3 │  .7176144             7   17.64713     0.0365
     4 │  .6538635             8   17.32777     0.0539
     5 │   .595776            12   16.87904     0.0784
     6 │  .5428489            14    16.3203     0.1089
     7 │  .4946237            15   15.74852     0.1401
     8 │  .4506827            18    15.2143     0.1693
  (output omitted)
    22 │  .1225221            52   12.02453     0.3435
  * 23 │  .1116376            59   12.02148     0.3436
    24 │    .10172            62   12.02571     0.3434
    25 │  .0926835            71   12.03785     0.3427
    26 │  .0844497            76    12.0626     0.3414
    27 │  .0769474            80   12.09713     0.3395
    27 │  .0769474            80   12.09713     0.3395
───────┴──────────────────────────────────────────────
* lambda selected by cross-validation.
```

For a sensitivity analysis that uses lassoselect after lassoknots, see [LASSO] **lassoselect**.

◁

Stored results

`lassoknots` stores the following in `r()`:

Matrices

`r(table)`	matrix containing the values displayed

Methods and formulas

Methods and formulas are presented under the following headings:

Overview

All the reported statistics depend on the p-dimensional coefficient vector $\widehat{\boldsymbol{\beta}}_\lambda$, which is the penalized estimate of $\boldsymbol{\beta}$ for given penalty value λ.

We present the formulas in the context of `lasso`, but formulas for `elasticnet` and `sqrtlasso` are the same, although the context would have some subtle differences that we can safely ignore.

Statistics that measure the size of the coefficient vector

Option `display(nonzero)` displays the number of nonzero coefficients, which is given by

$$\texttt{nonzero} = \sum_{j=1}^{p} d_j$$

$$d_j = \begin{cases} 1 & \text{if } \widehat{\beta}_{\lambda,j} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Option `display(l1)` displays the sum of the absolute values of the coefficients, which is known as the ℓ_1-norm:

$$\texttt{l1} = \sum_{j=1}^{p} |\widehat{\beta}_{\lambda,j}|$$

Option `display(l2)` displays the sum of the squared values of the coefficients, which is the square of the ℓ_2-norm:

$$\texttt{l2} = \sum_{j=1}^{p} \widehat{\beta}_{\lambda,j}^2$$

Statistics that measure fit

All statistics that measure fit are functions of the observation-level contributions of either the squared prediction error, spe_i, or the log likelihood, ℓ_i.

The contribution of observation i to a statistic can be calculated using a single-sample calculation or using CV. The CV version estimates the out-of-sample equivalent. The single-sample versions are in-sample measures that do not reliably estimate their out-of-sample equivalents.

CV measures of fit

When CV is performed, CV versions of spe_i and ℓ_i are available. Here is how we compute these observation-level quantities.

1. The data are partitioned into K folds.
2. For each value of λ,
   a. the coefficients are estimated on the observations not in fold k using λ.
   b. for each observation i in fold k, the fit measures spe_i and ℓ_i are computed using the penalized coefficient estimates.

Single-sample measures of fit

The single-sample measures of fit are computed as follows.

1. For each value of λ,
   a. the coefficients are estimated on all the observations using λ.
   b. for each observation i the fit measures spe_i and ℓ_i are computed using the penalized coefficient estimates.

Deviance formulas

The CV version of ℓ_i is used in the formulas for `cvmd` and `cvdevratio`. The single-sample version of ℓ_i is used in the formula for `devratio`.

For all models, the deviance, D_i, for the ith observation is given by

$$D_i = -2(\ell_i - \ell_{\text{saturated}})$$

where ℓ_i is the value of the log-likelihood function at observation i, and $\ell_{\text{saturated}}$ is the value of the saturated log-likelihood function. Formulas for the ℓ_i and for the $\ell_{\text{saturated}}$ are given below. The penalized coefficient estimates are used in these computations.

The mean deviance $\overline{D}$ is given by

$$\overline{D} = \frac{1}{N}\sum_{i=1}^{N} D_i$$

The formula for the deviance ratio D_2 is

$$D_2 = 1 - \frac{\overline{D}}{D_{\text{null}}}$$

where the D_{null} is the null deviance and is given by

$$D_{\text{null}} = \frac{1}{N}\sum_{i=1}^{N} -2(\ell_{0,i} - \ell_{\text{saturated}})$$

and $\ell_{0,i}$ is the ith observation of the log likelihood from the model that includes only a constant term.

Saturated log likelihood

For linear, logit, and probit models, the log-likelihood function of the saturated model is zero. For the Poisson model,

$$\ell_{\text{saturated}} = \frac{1}{N}\sum_{i=1}^{N}(-y_i + y_i \ln y_i)$$

For the Cox model,

$$\ell_{\text{saturated}} = -\frac{1}{N}\sum_{j=1}^{N_f} d_j \log\left(d_j\right)$$

where j indexes the ordered failure times $t_{(j)}$, $j = 1, \dots, N_f$; D_j is the set of observations that fail at $t_{(j)}$; and d_j is the number of observations in D_j.

Prediction error formulas

These formulas are used only for linear models. The squared prediction error for the ith observation is given by

$$\text{spe}_i = \left(y_i - \mathbf{x}_i\widehat{\boldsymbol{\beta}}_\lambda\right)^2$$

where y_i is the ith observation of the dependent variable and $\mathbf{x}_i\widehat{\boldsymbol{\beta}}_\lambda$ is the predicted mean of y_i conditional on $\mathbf{x}_i$.

For `cvmpe` and `osr2`, the CV version of spe_i is used. For `r2`, the single-sample version of spe_i is used.

R^2 is given by

$$R^2 = 1 - \frac{\text{MSE}}{\text{MSE}_{\text{null}}}$$

where the mean squared error (MSE) is given by

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N} \text{spe}_i$$

and the MSE of the null model is given by

$$\text{MSE}_{\text{null}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \overline{y})^2$$

where $\overline{y}$ is the sample average of y.

BIC formula

BIC is given by

$$\text{BIC} = -2\ell + k \ln N$$

where $\ell = \sum_{i=1}^{N} \ell_i$, $k = \texttt{nonzero} + 1$ is the number of coefficients in the model including the constant term, and each ℓ_i is always calculated using the single-sample methods.

References

Hastie, T. J., R. J. Tibshirani, and J. H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* 2nd ed. New York: Springer. https://doi.org/10.1007/978-0-387-84858-7.

Zhang, Y., R. Li, and C.-L. Tsai. 2010. Regularization parameter selections via generalized information criterion. *Journal of the American Statistical Association* 105: 312–323. https://doi.org/10.1198/jasa.2009.tm08013.

Also see

[LASSO] **lasso** — Lasso for prediction and model selection

[LASSO] **lasso fitting** — The process (in a nutshell) of fitting lasso models

[LASSO] **lassocoef** — Display coefficients after lasso estimation results

[LASSO] **lassoselect** — Select lambda after lasso

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[CAUSAL] **telasso postestimation** — Postestimation tools for telasso

lasso options — Lasso options for inferential models

Description Syntax Options Remarks and examples Reference Also see

Description

This entry describes the options that control the lassos, either individually or globally, in the ds, po, and xpo estimation commands.

For an introduction to lasso inferential models, see [LASSO] **Lasso inference intro**.

For examples of the ds, po, and xpo estimation commands and the use of these options, see [LASSO] **Inference examples**.

Syntax

lasso_inference_cmd ... [, ... *options*]

lasso_inference_cmd is one of dslogit, dspoisson, dsregress, poivregress, pologit, popoisson, poregress, xpoivregress, xpologit, xpopoisson, or xporegress.

options	Description
Model	
selection(plugin)	select λ^* using a plugin iterative formula for all lassos; the default
selection(cv)	select λ^* using cross-validation (CV) for all lassos
selection(adaptive)	select λ^* using adaptive lasso for all lassos
selection(bic)	select λ^* using Bayesian information criterion (BIC) for all lassos
sqrtlasso	fit square-root lassos instead of regular lassos
Advanced	
lasso(*varlist*, *lasso_options*)	specify options for lassos for variables in *varlist*
sqrtlasso(*varlist*, *lasso_options*)	specify options for square-root lassos for variables in *varlist*

lasso_options	Description
selection(*sel_method*)	selection method to select an optimal value of the lasso penalty parameter λ^* from the set of possible λ's
grid(#$_g$ [, ratio(#) min(#)])	specify the set of possible λ's using a logarithmic grid with #$_g$ grid points
stop(#)	tolerance for stopping the iteration over the λ grid early
cvtolerance(#)	tolerance for identification of the CV function minimum
bictolerance(#)	tolerance for identification of the BIC function minimum
tolerance(#)	convergence tolerance for coefficients based on their values
dtolerance(#)	convergence tolerance for coefficients based on deviance

sel_method	Description
`plugin` [, *plugin_opts*]	select λ^* using a plugin iterative formula; the default
`cv` [, *cv_opts*]	select λ^* using CV
`adaptive` [, *adapt_opts cv_opts*]	select λ^* using an adaptive lasso; only available for `lasso()`
`bic` [, *bic_opts*]	select λ^* using BIC

plugin_opts	Description
`heteroskedastic`	assume model errors are heteroskedastic; the default
`homoskedastic`	assume model errors are homoskedastic

cv_opts	Description
`folds(#)`	use # folds for CV
`alllambdas`	fit models for all λ's in the grid or until the `stop(#)` tolerance is reached; by default, the CV function is calculated sequentially by λ, and estimation stops when a minimum is identified
`serule`	use the one-standard-error rule to select λ^*
`stopok`	when the CV function does not have an identified minimum and the `stop(#)` stopping criterion for λ was reached at λ_{stop}, set the selected λ^* to be λ_{stop}; the default
`strict`	do not select λ^* when the CV function does not have an identified minimum; this is a stricter alternative to the default `stopok`
`gridminok`	when the CV function does not have an identified minimum and the `stop(#)` stopping criterion for λ was not reached, set the selected λ^* to be the minimum of the λ grid, λ_{gmin}; this is a looser alternative to the default `stopok` and is rarely used

adapt_opts	Description
`steps(#)`	use # adaptive steps (counting the initial lasso as step 1)
`unpenalized`	use the unpenalized estimator to construct initial weights
`ridge`	use the ridge estimator to construct initial weights
`power(#)`	raise weights to the #th power

bic_opts	Description
`alllambdas`	fit models for all λ's in the grid or until the `stop(#)` tolerance is reached; by default, the BIC function is calculated sequentially by λ, and estimation stops when a minimum is identified
`stopok`	when the BIC function does not have an identified minimum and the `stop(#)` stopping criterion for λ was reached at λ_{stop}, set the selected λ^* to be λ_{stop}; the default
`strict`	do not select λ^* when the BIC function does not have an identified minimum; this is a stricter alternative to the default `stopok`
`gridminok`	when the BIC function does not have an identified minimum and the `stop(#)` stopping criterion for λ was not reached, set the selected λ^* to be the minimum of the λ grid, λ_{gmin}; this is a looser alternative to the default `stopok` and is rarely used
`postselection`	use postselection coefficients to compute BIC

Options

Model

`selection(plugin | cv | adaptive | bic)` is a global option that specifies that all lassos use the given selection method. It is the same as specifying `lasso(*, selection(plugin | cv | adaptive | bic))`. The default is `selection(plugin)`. That is, not specifying this option implies a global `selection(plugin)` for all lassos. This global form of the option does not allow suboptions. To specify suboptions, use the `lasso()` or `sqrtlasso()` option described below.

`sqrtlasso` is a global option that specifies that all lassos be square-root lassos. It is the same as specifying `sqrtlasso(*)`, except for logit and Poisson models. For logit and Poisson models, it is the same as `sqrtlasso(`*varsofinterest*`)`, where *varsofinterest* are all the variables that have lassos excluding the dependent variable. This global form of the option does not allow suboptions. To specify suboptions, use the `sqrtlasso()` option described below.

Advanced

`lasso(`*varlist*`,` *lasso_options*`)` and `sqrtlasso(`*varlist*`,` *lasso_options*`)` let you set different options for different lassos and square-root lassos. These options also let you specify advanced options for all lassos and all square-root lassos. The `lasso()` and `sqrtlasso()` options override the global options `selection(plugin | cv | adaptive)` and `sqrtlasso` for the lassos for the specified variables. If `lasso(`*varlist*`,` *lasso_options*`)` or `sqrtlasso(`*varlist*`,` *lasso_options*`)` does not contain a `selection()` specification as part of *lasso_options*, then the global option for `selection()` is assumed.

`lasso(`*varlist*`,` *lasso_options*`)` specifies that the variables in *varlist* be fit using lasso with the selection method, set of possible λ's, and convergence criteria determined by *lasso_options*.

`sqrtlasso(`*varlist*`,` *lasso_options*`)` specifies that the variables in *varlist* be fit using square-root lasso with the selection method, set of possible λ's, and convergence criteria determined by *lasso_options*.

For `lasso()` and `sqrtlasso()`, *varlist* consists of one or more variables from *depvar*, the dependent variable, or *varsofinterest*, the variables of interest. To specify options for all lassos, you may use `*` or `_all` to specify *depvar* and all *varsofinterest*.

For models with endogeneity, namely, poivregress and xpoivregress models, lassos are done for *depvar*, the exogenous variables, *exovars*, and the endogenous variables, *endovars*. Any of these variables can be specified in the lasso() option. All of them can be specified using * or _all.

The lasso() and sqrtlasso() options are repeatable as long as different variables are given in each specification of lasso() and sqrtlasso(). The type of lasso for any *depvar* or *varsofinterest* (or *exovars* or *endovars*) not specified in any lasso() or sqrtlasso() option is determined by the global lasso options described above.

For all lasso inferential commands, linear lassos are done for each of the *varsofinterest* (or *exovars* and *endovars*). For linear models, linear lassos are also done for *depvar*. For logit models, however, logit lassos are done for *depvar*. For Poisson models, Poisson lassos are done for *depvar*. Square-root lassos are linear models, so sqrtlasso(*depvar*, ...) cannot be specified for the dependent variable in logit and Poisson models. For the same reason, sqrtlasso(*, ...) and sqrtlasso(_all, ...) cannot be specified for logit and Poisson models. For logit and Poisson models, you must specify sqrtlasso(*varsofinterest*, ...) to set options for square-root lassos and specify lasso(*depvar*, ...) to set options for the logit or Poisson lasso for *depvar*.

Suboptions for lasso( ) and sqrtlasso( )

selection(plugin [, heteroskedastic homoskedastic]) selects λ^* based on a "plugin" iterative formula dependent on the data. The plugin estimator calculates a value for λ^* that dominates the noise in the estimating equations, which ensures that the variables selected belong to the true model with high probability. See *Methods and formulas* in [LASSO] **lasso**.

selection(plugin) does not estimate coefficients for any other values of λ, so it does not require a λ grid, and none of the grid options apply. It is much faster than the other selection methods because estimation is done only for a single value of λ. It is an iterative procedure, however, and if the plugin is computing estimates for a small λ (which means many nonzero coefficients), the estimation can still be time consuming.

heteroskedastic assumes model errors are heteroskedastic. It is the default. Specifying selection(plugin) for linear lassos is equivalent to specifying selection(plugin, heteroskedastic). This suboption can be specified only for linear lassos. Hence, this suboption cannot be specified for *depvar* for logit and Poisson models, where *depvar* is the dependent variable. For these models, specify lasso(*depvar*, selection(plugin)) to have the logit or Poisson plugin formula used for the lasso for *depvar*. See *Methods and formulas* in [LASSO] **lasso**.

homoskedastic assumes model errors are homoskedastic. This suboption can be specified only for linear lassos. Hence, this suboption cannot be specified for *depvar* for logit and Poisson models, where *depvar* is the dependent variable.

selection(cv [, folds(#) alllambdas serule stopok strict gridminok]) selects λ^* to be the λ that gives the minimum of the CV function.

folds(#) specifies that CV with # folds be done. The default is folds(10).

alllambdas specifies that models be fit for all λ's in the grid or until the stop(#) tolerance is reached. By default, models are calculated sequentially from largest to smallest λ, and the CV function is calculated after each model is fit. If a minimum of the CV function is found, the computation ends at that point without evaluating additional smaller λ's.

alllambdas computes models for these additional smaller λ's. Because computation time is greater for smaller λ, specifying alllambdas may increase computation time manyfold. Specifying alllambdas is typically done only when a full plot of the CV function is wanted for assurance that a true minimum has been found. Regardless of whether alllambdas is specified, the selected λ^* will be the same.

serule selects λ^* based on the "one-standard-error rule" recommended by Hastie, Tibshirani, and Wainwright (2015, 13–14) instead of the λ that minimizes the CV function. The one-standard-error rule selects the largest λ for which the CV function is within a standard error of the minimum of the CV function.

stopok, strict, and gridminok specify what to do when the CV function does not have an identified minimum. A minimum is identified at λ^* when the CV function at both larger and smaller adjacent λ is greater than it is at λ^*. When the CV function has an identified minimum, stopok, strict, and gridminok all do the same thing: the selected λ^* is the λ that gives the minimum.

In some cases, however, the CV function declines monotonically as λ gets smaller and never rises to identify a minimum. When the CV function does not have an identified minimum, stopok and gridminok make alternative selections for λ^*, and strict makes no selection. You may specify only one of stopok, strict, or gridminok; stopok is the default if you do not specify one. With each of these suboptions, estimation results are always left in place, and alternative λ^* can be selected and evaluated.

stopok specifies that when the CV function does not have an identified minimum and the stop(#) stopping tolerance for λ was reached, the selected λ^* is λ_{stop}, the λ that met the stopping criterion. λ_{stop} is the smallest λ for which coefficients are estimated, and it is assumed that λ_{stop} has a CV function value close to the true minimum. When no minimum is identified and the stop(#) criterion is not met, an error is issued.

strict requires the CV function to have an identified minimum. If it does not, an error is issued.

gridminok is a rarely used suboption that specifies that when the CV function has no identified minimum and the stop(#) stopping criterion was not met, λ_{gmin}, the minimum of the λ grid, is the selected λ^*.

The gridminok selection criterion is looser than the default stopok, which is looser than strict. With strict, only an identified minimum is selected. With stopok, either the identified minimum or λ_{stop} is selected. With gridminok, either the identified minimum or λ_{stop} or λ_{gmin} is selected, in this order.

selection(adaptive [, steps(#) unpenalized ridge power(#) *cv_options*]) can be specified only as a suboption for lasso(). It cannot be specified as a suboption for sqrtlasso(). It selects λ^* using the adaptive lasso selection method. It consists of multiple lassos with each lasso step using CV. Variables with zero coefficients are discarded after each successive lasso, and variables with nonzero coefficients are given penalty weights designed to drive small coefficient estimates to zero in the next step. Hence, the final model typically has fewer nonzero coefficients than a single lasso.

selection(bic [, *bic_opts*]) selects λ^* to be the λ that gives the minimum of the BIC function.

bic_opts are alllambdas, stopok, strict, gridminok, and postselection.

alllambdas specifies that models be fit for all λ's in the grid or until the stop(#) tolerance is reached. By default, models are calculated sequentially from largest to smallest λ, and the BIC function is calculated after each model is fit. If a minimum of the BIC function is found, the computation ends at that point without evaluating additional smaller λ's.

alllambdas computes models for these additional smaller λ's. Because computation time is greater for smaller λ, specifying alllambdas may increase computation time manyfold. Specifying alllambdas is typically done only when a full plot of the BIC function is wanted for assurance that a true minimum has been found. Regardless of whether alllambdas is specified, the selected λ^* will be the same.

stopok, strict, and gridminok specify what to do when the BIC function does not have an identified minimum. A minimum is identified at λ^* when the BIC function at both larger and smaller adjacent λ's is greater than it is at λ^*. When the BIC function has an identified minimum, these options all do the same thing: the selected λ^* is the λ that gives the minimum. In some cases, however, the BIC function declines monotonically as λ gets smaller and never rises to identify a minimum. When the BIC function does not have an identified minimum, stopok and gridminok make alternative selections for λ^*, and strict makes no selection. You may specify only one of stopok, strict, or gridminok; stopok is the default if you do not specify one. With each of these options, estimation results are always left in place, and alternative λ^* can be selected and evaluated.

stopok specifies that when the BIC function does not have an identified minimum and the stop(#) stopping tolerance for λ was reached, the selected λ^* is λ_{stop}, the λ that met the stopping criterion. λ_{stop} is the smallest λ for which coefficients are estimated, and it is assumed that λ_{stop} has a BIC function value close to the true minimum. When no minimum is identified and the stop(#) criterion is not met, an error is issued.

strict requires the BIC function to have an identified minimum, and if not, an error is issued.

gridminok is a rarely used option that specifies that when the BIC function has no identified minimum and the stop(#) stopping criterion was not met, then λ_{gmin}, the minimum of the λ grid, is the selected λ^*.

The gridminok selection criterion is looser than the default stopok, which is looser than strict. With strict, only an identified minimum is selected. With stopok, either the identified minimum or λ_{stop} is selected. With gridminok, either the identified minimum or λ_{stop} or λ_{gmin} is selected, in this order.

postselection specifies to use the postselection coefficients to compute the BIC function. By default, the penalized coefficients are used.

steps(#) specifies that adaptive lasso with # lassos be done. By default, # $=$ 2. That is, two lassos are run. After the first lasso estimation, terms with nonzero coefficients β_i are given penalty weights equal to $1/|\beta_i|$, terms with zero coefficients are omitted, and a second lasso is estimated. Terms with small coefficients are given large weights, making it more likely that small coefficients become zero in the second lasso. Setting # $>$ 2 can produce more parsimonious models. See *Methods and formulas* in [LASSO] **lasso**

unpenalized specifies that the adaptive lasso use the unpenalized estimator to construct the initial weights in the first lasso. unpenalized is useful when CV cannot find a minimum. unpenalized cannot be specified with ridge.

ridge specifies that the adaptive lasso use the ridge estimator to construct the initial weights in the first lasso. ridge cannot be specified with unpenalized.

power(#) specifies that the adaptive lasso raise the weights to the #th power. The default power is 1. The specified power must be in the interval $[0.25, 2]$.

cv_options are all the suboptions that can be specified for `selection(cv)`, namely, `folds(#)`, `alllambdas`, `serule`, `stopok`, `strict`, and `gridminok`. The suboptions `alllambdas`, `strict`, and `gridminok` apply only to the first lasso estimated. For second and subsequent lassos, `gridminok` is the default. When `ridge` is specified, `gridminok` is automatically used for the first lasso.

`grid(`$\#_g$ [`, ratio(#) min(#)`]`)` specifies the set of possible λ's using a logarithmic grid with $\#_g$ grid points.

$\#_g$ is the number of grid points for λ. The default is $\#_g = 100$. The grid is logarithmic with the ith grid point ($i = 1, \dots, n = \#_g$) given by $\ln \lambda_i = [(i-1)/(n-1)] \ln r + \ln \lambda_{\text{gmax}}$, where $\lambda_{\text{gmax}} = \lambda_1$ is the maximum, $\lambda_{\text{gmin}} = \lambda_n =$ `min(#)` is the minimum, and $r = \lambda_{\text{gmin}}/\lambda_{\text{gmax}} =$ `ratio(#)` is the ratio of the minimum to the maximum.

`ratio(#)` specifies $\lambda_{\text{gmin}}/\lambda_{\text{gmax}}$. The maximum of the grid, λ_{gmax}, is set to the smallest λ for which all the coefficients in the lasso are estimated to be zero (except the coefficients of the *alwaysvars*). λ_{gmin} is then set based on `ratio(#)`. When $p < N$, where p is the total number of *othervars* and *alwaysvars* (not including the constant term) and N is the number of observations, the default value of `ratio(#)` is 1e−4. When $p \geq N$, the default is 1e−2.

`min(#)` sets λ_{gmin}. By default, λ_{gmin} is based on `ratio(#)` and λ_{gmax}, which is computed from the data.

`stop(#)` specifies a tolerance that is the stopping criterion for the λ iterations. The default is 1e−5. This suboption does not apply when the selection method is `selection(plugin)`. Estimation starts with the maximum grid value, λ_{gmax}, and iterates toward the minimum grid value, λ_{gmin}. When the relative difference in the deviance produced by two adjacent λ grid values is less than `stop(#)`, the iteration stops and no smaller λ's are evaluated. The value of λ that meets this tolerance is denoted by λ_{stop}. Typically, this stopping criterion is met before the iteration reaches λ_{gmin}.

Setting `stop(#)` to a larger value means that iterations are stopped earlier at a larger λ_{stop}. To produce coefficient estimates for all values of the λ grid, `stop(0)` can be specified. Note, however, that computations for small λ's can be extremely time consuming. In terms of time, when using `selection(cv)`, `selection(adaptive)`, or `selection(bic)`, the optimal value of `stop(#)` is the largest value that allows estimates for just enough λ's to be computed to identify the minimum of the CV or BIC function. When setting `stop(#)` to larger values, be aware of the consequences of the default λ^* selection procedure given by the default `stopok`. You may want to override the `stopok` behavior by using `strict`.

`cvtolerance(#)` is a rarely used option that changes the tolerance for identifying the minimum CV function. For linear models, a minimum is identified when the CV function rises above a nominal minimum for at least three smaller λ's with a relative difference in the CV function greater than #. For nonlinear models, at least five smaller λ's are required. The default is 1e−3. Setting # to a bigger value makes a stricter criterion for identifying a minimum and brings more assurance that a declared minimum is a true minimum, but it also means that models may need to be fit for additional smaller λ, which can be time consuming. See *Methods and formulas* for [LASSO] **lasso** for more information about this tolerance and the other tolerances.

`bictolerance(#)` is a rarely used option that changes the tolerance for identifying the minimum BIC function. A minimum is identified when the BIC function rises above a nominal minimum for at least two smaller λ's with a relative difference in the BIC function greater than #. The default is 1e−2. Setting # to a bigger value makes a stricter criterion for identifying a minimum and brings more assurance that a declared minimum is a true minimum, but it also means that models may need to be fit for additional smaller λ, which can be time consuming. See *Methods and formulas* in [LASSO] **lasso** for more information about this tolerance and the other tolerances.

`tolerance(#)` is a rarely used option that specifies the convergence tolerance for the coefficients. Convergence is achieved when the relative change in each coefficient is less than this tolerance. The default is `tolerance(1e-7)`.

`dtolerance(#)` is a rarely used option that changes the convergence criterion for the coefficients. When `dtolerance(#)` is specified, the convergence criterion is based on the change in deviance instead of the change in the values of coefficient estimates. Convergence is declared when the relative change in the deviance is less than #. More-accurate coefficient estimates are typically achieved by not specifying this option and instead using the default `tolerance(1e-7)` criterion or specifying a smaller value for `tolerance(#)`.

Remarks and examples

All the options shown here may seem overwhelming. However, you will likely never need to use many of them.

You would typically use the global options to change the selection method for each of the lassos performed by one of the lasso inference commands. For example, you can specify `selection(cv)`, `selection(adaptive)`, or `selection(bic)` to change the selection method globally from the default `selection(plugin)`.

Sometimes, CV fails to identify a minimum of the CV function and so fails to select λ^*; thus, the inferential command fails. Lasso inference postestimation commands provide tools to see what happened. Then it may be possible to set options so that an acceptable λ^* is selected. Of course, in many cases, the issue is not with the computation but rather with model specification or simply not having enough data.

To understand the `selection(cv)`, `selection(adaptive)`, and `selection(bic)` selection methods and how to set options to control them, you should first become familiar with lasso for prediction and model selection.

Notice, however, that options for the `lasso` and `sqrtlasso` commands are specified slightly differently than they are when used as suboptions for lasso inference commands. For instance, with `lasso`, you might specify `selection(cv, folds(20))`. With `dsregress` or one of the other inference commands, you would specify `lasso(*, selection(cv, folds(20)))` to specify that CV with 20 folds be used to select λ^* for each lasso.

Read [LASSO] **lasso** and [LASSO] **lasso fitting** to learn about the lasso options in greater detail.

Reference

Hastie, T. J., R. J. Tibshirani, and M. Wainwright. 2015. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Boca Raton, FL: CRC Press. https://doi.org/10.1201/b18401.

Also see

[LASSO] **Lasso intro** — Introduction to lasso

[LASSO] **Lasso inference intro** — Introduction to inferential lasso models

[LASSO] **lasso** — Lasso for prediction and model selection

[LASSO] **lasso fitting** — The process (in a nutshell) of fitting lasso models

lassoselect — Select lambda after lasso

Description	Quick start	Menu	Syntax	Options
Remarks and examples	Stored results	Also see		

Description

lassoselect allows the user to select a different λ^* after lasso and sqrtlasso when the selection method was selection(cv), selection(adaptive), selection(bic), or selection(none).

After elasticnet, the user can select a different (α^*, λ^*) pair.

When the telasso, ds, po, and xpo commands fit models using selection(cv), selection(adaptive), or selection(bic) ([LASSO] **lasso options**), lassoselect can be used to select a different λ^* for a particular lasso.

Quick start

After lasso with selection(cv), change the selected λ^* to that with ID = 52

```
lassoselect id = 52
```

Same as above, but change the selected λ^* to the λ closest to 0.01

```
lassoselect lambda = 0.01
```

After elasticnet, change the selected (α^*, λ^*) to $(0.5, 0.267345)$

```
lassoselect alpha = 0.5 lambda = 0.267345
```

After dsregress with selection(adaptive), change the selected λ^* to 1.65278 for the adaptive lasso for the variable y

```
lassoselect lambda = 1.65278, for(y)
```

After poivregress with selection(bic), change the selected λ^* to the λ closest to 0.7 for the lasso for the prediction of the variable income

```
lassoselect lambda = 0.7, for(pred(income))
```

After xporegress with selection(cv) and resample, change the selected λ^* to 0.234189 for the lasso for the variable x26 for the 5th cross-fit fold in the 9th resample

```
lassoselect lambda = 0.234189, for(x26) xfold(5) resample(9)
```

After telasso with selection(cv), change the selected λ^* to the λ closest to 0.7 for the lasso for the outcome variable y at treatment level 1

```
lassoselect lambda = 0.7, for(y) tlevel(1)
```

Menu

Statistics > Postestimation

Syntax

After `lasso`, `sqrtlasso`, *and* `elasticnet`

`lassoselect id = #`

After `lasso` *and* `sqrtlasso`

`lassoselect lambda = #`

After `elasticnet`

`lassoselect alpha = # lambda = #`

After `ds` *and* `po` *with* `selection(cv)` *or* `selection(adaptive)`

`lassoselect { id | lambda } = # , for(`*varspec*`)`

After `xpo` *without* `resample` *and with* `selection(cv)` *or* `selection(adaptive)`

`lassoselect { id | lambda } = # , for(`*varspec*`) xfold(#)`

After `xpo` *with* `resample` *and* `selection(cv)` *or* `selection(adaptive)`

`lassoselect { id | lambda } = # , for(`*varspec*`) xfold(#) resample(#)`

After `telasso` *for the outcome variable and with* `selection(cv)` *or* `selection(adaptive)`

`lassoselect { id | lambda } = #, for(`*varspec*`) tlevel(#)`

After `telasso` *for the treatment variable and with* `selection(cv)` *or* `selection(adaptive)`

`lassoselect { id | lambda } = #, for(`*varspec*`)`

After `telasso` *for the outcome variable with cross-fitting but without* `resample` *and with* `selection(cv)` *or* `selection(adaptive)`

`lassoselect { id | lambda } = #, for(`*varspec*`) tlevel(#) xfold(#)`

After `telasso` *for the treatment variable with cross-fitting but without* `resample`

`lassoselect { id | lambda } = #, for(`*varspec*`) xfold(#)`

After `telasso` *for the outcome variable with cross-fitting and* `resample` *and with* `selection(cv)` *or* `selection(adaptive)`

`lassoselect { id | lambda } = #, for(`*varspec*`) tlevel(#) xfold(#) resample(#)`

After `telasso` *for the treatment variable with cross-fitting and* `resample` *and with* `selection(cv)` *or* `selection(adaptive)`

`lassoselect { id | lambda } = #, for(`*varspec*`) xfold(#) resample(#)`

varspec is *varname*, except after poivregress and xpoivregress, when it is either *varname* or pred(*varname*).

options	Description
* for(*varspec*)	lasso for *varspec*; telasso, ds, po, and xpo commands only
* xfold(#)	lasso for the #th cross-fit fold; xpo commands and telasso with xfolds only
* resample(#)	lasso for the #th resample; xpo commands and telasso with resample only
* tlevel(#)	lasso for the outcome model with the treatment level #; telasso only

*for(*varspec*) is required for all ds, po, and xpo commands and for telasso.
xfold(#) is required for all xpo commands and for telasso when the option xfolds(#) was specified.
resample(#) is required for xpo and for telasso when the option resample(#) was specified.
tlevel(#) is required for the outcome model in telasso.
collect is allowed; see **[U] 11.1.10 Prefix commands**.

Options

for(*varspec*) specifies a particular lasso after telasso or after a ds, po, or xpo estimation command fit using the option selection(cv), selection(adaptive), or selection(bic). For all commands except poivregress and xpoivregress, *varspec* is always *varname*.

For the ds, po, and xpo commands except poivregress and xpoivregress, *varspec* is either *depvar*, the dependent variable, or one of *varsofinterest* for which inference is done.

For poivregress and xpoivregress, *varspec* is either *varname* or pred(*varname*). The lasso for *depvar* is specified with its *varname*. Each of the endogenous variables have two lassos, specified by *varname* and pred(*varname*). The exogenous variables of interest each have only one lasso, and it is specified by pred(*varname*).

For telasso, *varspec* is either the outcome variable or the treatment variable.

This option is required after telasso and after the ds, po, and xpo commands.

xfold(#) specifies a particular lasso after an xpo estimation command or after telasso when the option xfolds(#) was specified. For each variable to be fit with a lasso, K lassos are done, one for each cross-fit fold, where K is the number of folds. This option specifies which fold, where $\# = 1, 2, \ldots, K$. xfold(#) is required after an xpo command and after telasso when the option xfolds(#) was specified.

resample(#) specifies a particular lasso after an xpo estimation command or after telasso fit using the option resample(#). For each variable to be fit with a lasso, $R \times K$ lassos are done, where R is the number of resamples and K is the number of cross-fitting folds. This option specifies which resample, where $\# = 1, 2, \ldots, R$. resample(#), along with xfold(#), is required after an xpo command and after telasso with resampling.

tlevel(#) specifies the lasso for the outcome variable at the specified treatment level after telasso. This option is required to refer to the outcome model after telasso.

Remarks and examples

▷ Example 1: lasso linear

Here is an example using `lasso` from [LASSO] **lasso examples**. We load the data and make the `vl` variable lists active.

```
. use https://www.stata-press.com/data/r19/fakesurvey_vl
(Fictitious survey data with vl)

. vl rebuild
Rebuilding vl macros ...
  (output omitted)
```

We want to evaluate our lasso predictions on a sample that we did not use to fit the lasso. So we randomly split our data into two samples of equal sizes. We will fit models on one, and we will use the other to test their predictions. We use `splitsample` to generate a variable indicating the two subsamples.

```
. set seed 1234

. splitsample, generate(sample) nsplit(2)

. label define svalues 1 "Training" 2 "Testing"

. label values sample svalues
```

We fit a lasso linear model on the first subsample.

```
. lasso linear q104 ($idemographics) $ifactors $vlcontinuous
> if sample == 1, rseed(1234)

10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .8978025   no. of nonzero coef. =   4
Folds: 1...5....10   CVF = 16.93341
  (output omitted)
Grid value 23:    lambda = .1159557   no. of nonzero coef. =  74
Folds: 1...5....10   CVF = 12.17933
... cross-validation complete ... minimum found

Lasso linear model                          No. of obs        =        458
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds   =         10

--------------------------------------------------------------------------
                  |                                No. of     Out-of-      CV mean
                  |                               nonzero      sample   prediction
               ID |     Description      lambda     coef.   R-squared        error
------------------+-------------------------------------------------------
                1 |    first lambda    .8978025         4      0.0147     16.93341
               18 |   lambda before    .1846342        42      0.2953     12.10991
             * 19 | selected lambda    .1682318        49      0.2968     12.08516
               20 |    lambda after    .1532866        55      0.2964     12.09189
               23 |     last lambda    .1159557        74      0.2913     12.17933
--------------------------------------------------------------------------
* lambda selected by cross-validation.
```

We store the results because we want to compare these results with other results later.

```
. estimates store lassocv
```

We run lassoknots with options to show the number of nonzero coefficients, estimates of out-of-sample R^2, and the Bayes information criterion (BIC).

```
. lassoknots, display(nonzero osr2 bic)

------------------------------------------------------------------
             |                  No. of     Out-of-
             |                 nonzero      sample
          ID |     lambda        coef.   R-squared             BIC
-------------+----------------------------------------------------
           1 |   .8978025            4      0.0147        2618.642
           2 |   .8180442            7      0.0236        2630.961
           3 |   .7453714            8      0.0421        2626.254
           4 |   .6791547            9      0.0635        2619.727
           5 |   .6188205           10      0.0857        2611.577
           6 |   .5638462           13      0.1110        2614.155
           8 |    .468115           14      0.1581        2588.189
           9 |   .4265289           16      0.1785        2584.638
          10 |   .3886373           18      0.1980        2580.891
          11 |   .3541118           22      0.2170        2588.984
          12 |   .3226535           26      0.2340        2596.792
          13 |   .2939899           27      0.2517        2586.521
          14 |   .2678726           28      0.2669        2578.211
          15 |   .2440755           32      0.2784        2589.632
          16 |   .2223925           35      0.2865        2593.753
          17 |   .2026358           37      0.2919        2592.923
          18 |   .1846342           42      0.2953        2609.975
        * 19 |   .1682318           49      0.2968        2639.437
          20 |   .1532866           55      0.2964        2663.451
          21 |    .139669           62      0.2952        2693.929
          22 |   .1272612           66      0.2934        2707.174
          23 |   .1159557           74      0.2913        2744.508
------------------------------------------------------------------
* lambda selected by cross-validation.
```

Research indicates that under certain conditions, selecting the λ that minimizes the BIC gives good predictions. See *BIC* in [LASSO] **lassoknots**.

Here the λ with ID $=$ 14 gives the minimum value of the BIC. Let's select it.

```
. lassoselect id = 14
ID = 14  lambda = .2678726 selected
```

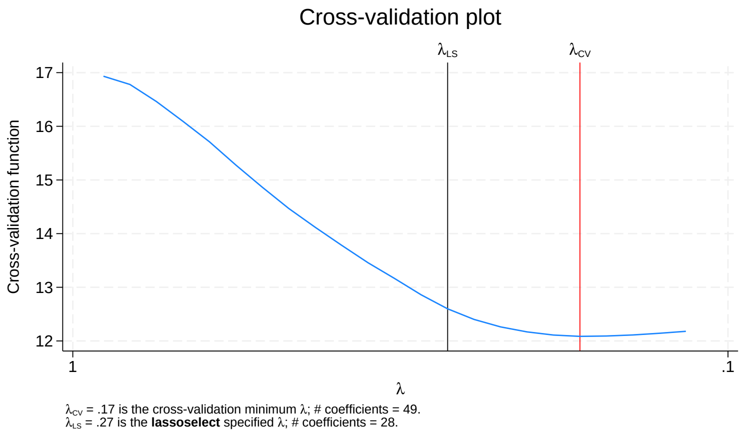
When lassoselect runs, it changes the current estimation results to correspond with the selected lambda. It is almost the same as running another estimation command and wiping out the old estimation results. We say "almost" because it is easy to change λ^* back to what it was originally. We stored our earlier results knowing lassoselect was going to do this.

Let's store the new results from lassoselect.

```
. estimates store lassosel
```

We plot the CV function with the new selected λ^* marked along with the λ selected by cross-validation—the λ that gives the minimum of the CV function.

```
. cvplot
```



The CV function is curving upward at the value of the new selected λ^*. Alternative λ^*'s in a region where the CV function is still relatively flat are sometimes selected, but that is not the case here.

The real test is to see how well it does for out-of-sample prediction compared with the original λ^*. We run `lassogof` to do this.

```
. lassogof lassocv lassosel, over(sample) postselection
Postselection coefficients
─────────────────────────────────────────────────────────────
Name             sample │         MSE    R-squared        Obs
────────────────────────┼────────────────────────────────────
lassocv                 │
               Training │    8.652771       0.5065        503
                Testing │    14.58354       0.2658        493
────────────────────────┼────────────────────────────────────
lassosel                │
               Training │    9.740229       0.4421        508
                Testing │    13.44496       0.3168        503
─────────────────────────────────────────────────────────────
```

The model for λ^* that minimized the BIC did considerably better on out-of-sample prediction than the model for λ^* that minimized the CV function. In-sample prediction was better for the λ^* that minimized the CV function. That is expected because that model contains more variables. But it appears these extra variables were mostly fitting noise, and that hurt the model's out-of-sample predictive ability.

◁

▷ Example 2: dsregress

`lassoselect` can be used after the `ds`, `po`, and `xpo` commands when they are fit using `selection(cv)` or `selection(adaptive)`. See [LASSO] **lasso options**.

We load the data used in [LASSO] **lasso examples**. See that entry for details about the data.

```
. use https://www.stata-press.com/data/r19/fakesurvey_vl, clear
(Fictitious survey data with vl)
. vl rebuild
Rebuilding vl macros ...
  (output omitted)
```

We are going to fit a `dsregress` model with `q104` as our dependent variable and variables of interest `q41` and `q22`. These variables of interest are currently in the variable lists `factors` and `vlcontinuous`, which we will use to specify the control variables. So we need to move them out of these variable lists.

```
. vl modify factors = factors - (q41)
note: 1 variable removed from $factors.
. vl move (q22) vlother
note: 1 variable specified and 1 variable moved.
  (output omitted)
. vl rebuild
Rebuilding vl macros ...
  (output omitted)
```

After we moved the variables out of the variable lists, we typed `vl rebuild` to update the variable list `ifactors` created from `factors`. See [D] **vl** for details.

Before we fit our `dsregress` model using cross-validation, let's fit it using the default `selection(plugin)`.

```
. dsregress q104 i.q41 q22, controls(($idemographics) $ifactors $vlcontinuous)
Estimating lasso for q104 using plugin
Estimating lasso for 1bn.q41 using plugin
Estimating lasso for q22 using plugin
Double-selection linear model         Number of obs               =        914
                                      Number of controls          =        274
                                      Number of selected controls =         33
                                      Wald chi2(2)                =      18.72
                                      Prob > chi2                 =     0.0001

------------------------------------------------------------------------------
             |               Robust
        q104 | Coefficient  std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         q41 |
        Yes  |   .8410538   .2691082     3.13   0.002     .3136114    1.368496
         q22 |  -.0878443   .0310435    -2.83   0.005    -.1486884   -.0270001
------------------------------------------------------------------------------
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

We run lassoinfo to see how many nonzero coefficients were in each lasso fit by dsregress. It is a good idea to always run lassoinfo after any ds, po, or xpo command.

```
. lassoinfo

    Estimate: active
     Command: dsregress

                                                       No. of
                             Selection                selected
    Variable        Model       method     lambda    variables

        q104       linear       plugin   .1467287           18
     1bn.q41       linear       plugin   .1467287           16
         q22       linear       plugin   .1467287           15
```

We now run dsregress with selection(cv),

```
. dsregress q104 i.q41 q22,
> controls(($idemographics) $ifactors $vlcontinuous)
> selection(cv) rseed(1234)
Estimating lasso for q104 using cv
Estimating lasso for 1bn.q41 using cv
Estimating lasso for q22 using cv
Double-selection linear model         Number of obs               =        914
                                      Number of controls          =        274
                                      Number of selected controls =        123
                                      Wald chi2(2)                =      10.96
                                      Prob > chi2                 =     0.0042

                            Robust
        q104   Coefficient  std. err.      z    P>|z|     [95% conf. interval]

         q41
         Yes     .6003918   .2848483     2.11   0.035     .0420994    1.158684
         q22    -.0681067   .0306219    -2.22   0.026    -.1281246   -.0080888

Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

and then run lassoinfo.

```
. lassoinfo

    Estimate: active
     Command: dsregress

                                                                  No. of
                             Selection  Selection                selected
    Variable        Model       method  criterion     lambda    variables

        q104       linear           cv    CV min.   .1116376           63
     1bn.q41       linear           cv    CV min.   .0135958           68
         q22       linear           cv    CV min.   .1624043           49
```

The selection(cv) lassos selected considerably more variables than the selection(plugin) lassos. The CV lassos selected 63, 68, and 49 variables for the lassos, whereas the plugin lassos selected 18, 16, and 15 variables.

We are going to use `lassoselect` to change the selected λ^* for CV lassos to match the number of selected variables in the plugin lassos.

```
. lassoknots, display(nonzero cvmpe osr2) for(q104)

─────────────────────────────────────────────────────────
         │                  No. of     CV mean    Out-of-
         │                 nonzero       pred.     sample
      ID │    lambda         coef.       error  R-squared
─────────┼───────────────────────────────────────────────
       1 │   .864369             4     17.9727     0.0187
       2 │  .7875809             6    17.88282     0.0236
       3 │  .7176144             7    17.64713     0.0365
       4 │  .6538635             8    17.32777     0.0539
       5 │   .595776            12    16.87904     0.0784
       6 │  .5428489            14     16.3203     0.1089
       7 │  .4946237            15    15.74852     0.1401
       8 │  .4506827            18     15.2143     0.1693
  (output omitted)
      22 │  .1225221            52    12.02453     0.3435
    * 23 │  .1116376            59    12.02148     0.3436
      24 │    .10172            62    12.02571     0.3434
      25 │  .0926835            71    12.03785     0.3427
      26 │  .0844497            76     12.0626     0.3414
      27 │  .0769474            80    12.09713     0.3395
      27 │  .0769474            80    12.09713     0.3395
─────────────────────────────────────────────────────────
* lambda selected by cross-validation.
. lassoknots, display(nonzero cvmpe osr2) for(1bn.q41)

─────────────────────────────────────────────────────────
         │                  No. of     CV mean    Out-of-
         │                 nonzero       pred.     sample
      ID │    lambda         coef.       error  R-squared
─────────┼───────────────────────────────────────────────
       1 │  .1155307             4    .2509624    -0.0044
       2 │  .1052673             5     .248763     0.0044
       3 │  .0959156             8    .2442525     0.0224
       4 │  .0873947             9    .2388787     0.0439
       5 │  .0796308            11    .2328436     0.0681
       6 │  .0725566            12    .2262371     0.0945
      10 │  .0500105            15    .2076117     0.1691
      12 │  .0415196            16    .2020617     0.1913
  (output omitted)
      23 │  .0149214            61    .1898068     0.2403
    * 24 │  .0135958            64    .1895992     0.2412
      25 │   .012388            68    .1896789     0.2408
      26 │  .0112875            76    .1900733     0.2393
      27 │  .0102847            87     .190537     0.2374
      28 │  .0093711            94     .190995     0.2356
─────────────────────────────────────────────────────────
* lambda selected by cross-validation.
```

```
. lassoknots, display(nonzero cvmpe osr2) for(q22)

─────────────────────────────────────────────────────────────
             |                No. of     CV mean     Out-of-
             |               nonzero       pred.      sample
          ID |     lambda      coef.       error   R-squared
─────────────┼───────────────────────────────────────────────
           1 |   1.380036          4    22.19516      0.0403
           2 |   1.257437          6    21.66035      0.0634
           3 |    1.14573          7    21.01623      0.0913
           5 |   .9512051          8    19.70951      0.1478
           9 |   .6556288          9    18.04511      0.2197
          10 |   .5973845         10    17.74092      0.2329
          11 |   .5443145         11    17.41052      0.2472
          12 |   .4959591         13    17.09005      0.2610
          13 |   .4518995         15    16.78501      0.2742
  (output omitted)
          23 |   .1782385         39    14.93049      0.3544
        * 24 |   .1624043         45    14.92344      0.3547
          25 |   .1479767         55    14.93826      0.3541
          26 |   .1348309         67    14.94057      0.3540
          27 |   .1228529         70    14.93962      0.3540
          28 |    .111939         75    14.95101      0.3535
─────────────────────────────────────────────────────────────
* lambda selected by cross-validation.
```

When we look at the `lassoinfo` output for the plugin lassos, we see that the value of λ^* for each lasso was the same, namely, 0.1467287. This value does not match up with the same numbers of nonzero coefficients for the CV lassos in these knot tables.

The plugin estimator for λ^* uses estimated coefficient-level weights in its lassos. In theoretical terms, these coefficient-level weights put λ^* on the correct scale for covariate selection by normalizing the scores of the unpenalized estimator. In practical terms, these weights cause the effective scale of λ for `selection(plugin)` and `selection(cv)` to differ.

We select the λ^*'s for each CV lasso to match the number of nonzero coefficients of the plugin lassos.

```
. lassoselect id = 6, for(q104)
ID = 6  lambda = .5428489 selected

. lassoselect id = 6, for(1bn.q41)
ID = 6  lambda = .0725566 selected

. lassoselect id = 11, for(q22)
ID = 11  lambda = .5443145 selected
```

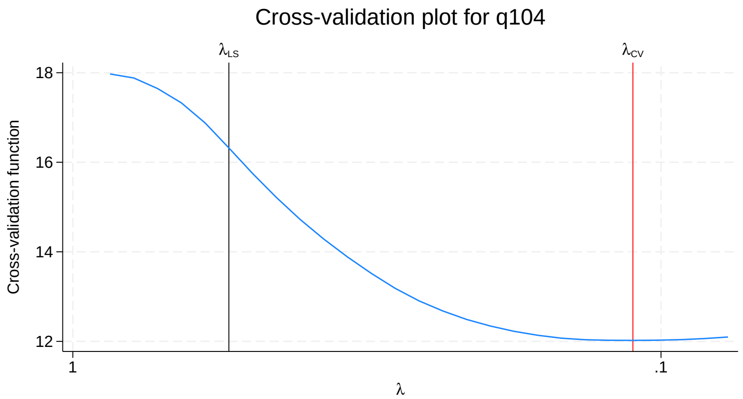
To update our dsregress model with these new λ^*'s, we rerun the command with the reestimate option. Then, we run lassoinfo to confirm that the lassos produced the same number of nonzero coefficients.

```
. dsregress, reestimate

Double-selection linear model         Number of obs               =        914
                                      Number of controls          =        274
                                      Number of selected controls =         33
                                      Wald chi2(2)                =      18.72
                                      Prob > chi2                 =     0.0001

─────────────┬────────────────────────────────────────────────────────────────
             │               Robust
        q104 │ Coefficient  std. err.      z    P>|z|     [95% conf. interval]
─────────────┼────────────────────────────────────────────────────────────────
         q41 │
        Yes  │   .8410538   .2691082     3.13   0.002     .3136114    1.368496
         q22 │  -.0878443   .0310435    -2.83   0.005    -.1486884   -.0270001
─────────────┴────────────────────────────────────────────────────────────────

Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.

. lassoinfo

    Estimate: active
     Command: dsregress

────────────┬─────────────────────────────────────────────────────────
            │                                                  No. of
            │            Selection  Selection                selected
    Variable│     Model     method  criterion     lambda    variables
────────────┼─────────────────────────────────────────────────────────
        q104│    linear       user       user   .5428489           18
    1bn.q41 │    linear       user       user   .0725566           16
         q22│    linear       user       user   .5443145           15
────────────┴─────────────────────────────────────────────────────────
```

These new dsregress results are exactly the same as the dsregress results produced with plugin lassos.

We can plot the CV function and see where the new λ^* falls. We do so for the lasso for the dependent variable q104.

```
. cvplot, for(q104)
```



λCV = .11 is the cross-validation minimum λ; # coefficients = 59.
λLS = .54 is the **lassoselect** specified λ; # coefficients = 14.

It may be that the plugin lassos underselected controls for this problem. Or it may be that the plugin lassos actually did fine and the CV lassos overselected controls. We might want to continue these sensitivity analyses and pick some λ^*'s intermediate between the plugin values and the CV values. Plugin selection and CV selection are not just two different numerical techniques, they are two different modeling techniques, each with a different set of assumptions. See [LASSO] **Inference requirements**.

◁

Stored results

lassoselect stores the following in r():

Macros	
r(varlist)	selected variables

Also see

[LASSO] **lasso postestimation** — Postestimation tools for lasso for prediction

[CAUSAL] **telasso postestimation** — Postestimation tools for telasso

poivregress — Partialing-out lasso instrumental-variables regression

Description

poviregress fits a lasso instrumental-variables linear regression model and reports coefficients along with standard errors, test statistics, and confidence intervals for specified covariates of interest. The covariates of interest may be endogenous or exogenous. The partialing-out method is used to estimate effects for these variables and to select from potential control variables and instruments to be included in the model.

Quick start

Estimate a coefficient for endogenous `d1` in a linear regression of `y` on `d1`, and include `x1` to `x100` as potential control variables and `z1` to `z100` as potential instruments to be selected by lassos

```
poivregress y (d1 = z1-z100), controls(x1-x100)
```

Same as above, and estimate the coefficient for the exogenous `d2`

```
poivregress y d2 (d1 = z1-z100), controls(x1-x100)
```

Use cross-validation (CV) instead of a plugin iterative formula to select the optimal λ^* in each lasso

```
poivregress y d2 (d1 = z1-z100), controls(x1-x100) selection(cv)
```

Same as above, and set a random-number seed for reproducibility

```
poivregress y d2 (d1 = z1-z100), controls(x1-x100) selection(cv) ///
   rseed(28)
```

Specify CV for the lasso for `y` only, with the stopping rule criterion turned off

```
poivregress y d2 (d1 = z1-z100), controls(x1-x100) ///
    lasso(y, selection(cv), stop(0))
```

Same as above, but apply the option to the lassos for `y`, `d2`, and `d1`

```
poivregress y d2 (d1 = z1-z100), controls(x1-x100) ///
    lasso(*, selection(cv), stop(0))
```

Compute lassos beyond the CV minimum to get full coefficient paths, knots, etc.

```
poivregress y d2 (d1 = z1-z100), controls(x1-x100) ///
    lasso(*, selection(cv, alllambdas))
```

Menu

Statistics > Lasso > Lasso inferential models > Continuous outcomes > Partialing-out IV model

Syntax

`poivregress` *depvar* [*exovars*] (*endovars* = *instrumvars*) [*if*] [*in*] ,
  `controls(`[(*alwaysvars*)] *othervars*`)` [*options*]

Coefficients and standard errors are estimated for the exogenous variables, *exovars*, and the endogenous variables, *endovars*. The set of instrumental variables, *instrumvars*, may be high dimensional.

options	Description
Model	
* `controls(`[(*alwaysvars*)] *othervars*`)`	*alwaysvars* and *othervars* are control variables for *depvar*, *exovars*, and *endovars*; *instrumvars* are an additional set of control variables that apply only to the *endovars*; *alwaysvars* are always included; lassos choose whether to include or exclude *othervars*
`selection(plugin)`	use a plugin iterative formula to select an optimal value of the lasso penalty parameter λ^* for each lasso; the default
`selection(cv)`	use CV to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(adaptive)`	use adaptive lasso to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(bic)`	use BIC to select an optimal value of the lasso penalty parameter λ^* for each lasso
`sqrtlasso`	use square-root lassos
`missingok`	after fitting lassos, ignore missing values in any *instrumvars* or *othervars* not selected, and include these observations in the final model
SE/Robust	
`vce(`*vcetype*`)`	*vcetype* may be `robust` (the default) or `cluster` *clustvar*
Reporting	
`level(#)`	set confidence level; default is `level(95)`
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
Optimization	
[`no`]`log`	display or suppress an iteration log
`verbose`	display a verbose iteration log
`rseed(#)`	set random-number seed
Advanced	
`lasso(`*varlist*, *lasso_options*`)`	specify options for the lassos for variables in *varlist*; may be repeated
`sqrtlasso(`*varlist*, *lasso_options*`)`	specify options for square-root lassos for variables in *varlist*; may be repeated
`reestimate`	refit the model after using `lassoselect` to select a different λ^*
`noheader`	do not display the header on the coefficient table
`coeflegend`	display legend instead of statistics

*`controls()` is required.

exovars, *endovars*, *instrumvars*, *alwaysvars*, and *othervars* may contain factor variables. Base levels of factor variables cannot be set for *instrumvars*, *alwaysvars*, and *othervars*. See [U] **11.4.3 Factor variables**.

`collect` is allowed; see [U] **11.1.10 Prefix commands**.

`reestimate`, `noheader`, and `coeflegend` do not appear in the dialog box.

See [U] **20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

`controls(`[`(`*alwaysvars*`)`] *othervars*`)` specifies the set of control variables, which control for omitted variables. Control variables are also known as confounding variables. *alwaysvars* are variables that are always to be included in lassos. *alwaysvars* are optional. *othervars* are variables that lassos will choose to include or exclude. The instrumental variables, *instrumvars*, are an additional set of control variables, but they apply only to the *endovars*. `controls()` is required.

`poivregress` fits lassos for *depvar* and each one of the *exovars* and *endovars*. The control variables for the lassos for *depvar* and *exovars* are *alwaysvars* (always included) and *othervars* (lasso will include or exclude). The control variables for lassos for *endovars* are *exovars* (always included), *alwaysvars* (always included), *instrumvars* (lasso will include or exclude), and *othervars* (lasso will include or exclude).

`selection(plugin` | `cv` | `adaptive` | `bic)` specifies the selection method for choosing an optimal value of the lasso penalty parameter λ^* for each lasso or square-root lasso estimation. Separate lassos are estimated for *depvar* and each variable in *varsofinterest*. Specifying `selection()` changes the selection method for all of these lassos. You can specify different selection methods for different lassos using the option `lasso()` or `sqrtlasso()`. When `lasso()` or `sqrtlasso()` is used to specify a different selection method for the lassos of some variables, they override the global setting made using `selection()` for the specified variables.

`selection(plugin)` is the default. It selects λ^* based on a "plugin" iterative formula dependent on the data. See [LASSO] **lasso options**.

`selection(cv)` selects the λ^* that gives the minimum of the CV function. See [LASSO] **lasso options**.

`selection(adaptive)` selects λ^* using the adaptive lasso selection method. It cannot be specified when `sqrtlasso` is specified. See [LASSO] **lasso options**.

`selection(bic)` selects the λ^* that gives the minimum of the BIC function. See [LASSO] **lasso options**.

`sqrtlasso` specifies that square-root lassos be done rather than regular lassos. The option `lasso()` can be used with `sqrtlasso` to specify that regular lasso be done for some variables, overriding the global `sqrtlasso` setting for these variables. See [LASSO] **lasso options**.

`missingok` specifies that, after fitting lassos, the estimation sample be redefined based on only the nonmissing observations of variables in the final model. In all cases, any observation with missing values for *depvar*, *exovars*, *endovars*, *instrumvars*, *alwaysvars*, and *othervars* is omitted from the estimation sample for the lassos. By default, the same sample is used for calculation of the coefficients of the *exovars* and *endovars* and their standard errors.

When missingok is specified, the initial estimation sample is the same as the default, but the sample used for the calculation of the coefficients of the *exovars* and *endovars* can be larger. Now observations with missing values for any *instrumvars* and *othervars* not selected will be added to the estimation sample (provided there are no missing values for any of the variables in the final model).

missingok may produce more efficient estimates when data are missing completely at random. It does, however, have the consequence that estimation samples can change when selected variables differ in models fit using different selection methods. That is, when *instrumvars* and *othervars* contain missing values, the estimation sample for a model fit using the default selection(plugin) will likely differ from the estimation sample for a model fit using, for example, selection(cv).

SE/Robust

vce(*vcetype*) specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (robust) and that allow for intragroup correlation (cluster *clustvar*); see [R] ***vce_option***.

When vce(cluster *clustvar*) is specified, all lassos also account for clustering. For each lasso, this affects how the log-likelihood function is computed and how the sample is split in cross-validation; see *Methods and formulas* in [LASSO] **lasso**. Specifying vce(cluster *clustvar*) may lead to different selected controls and therefore to different point estimates for your variable of interest when compared to the estimation that ignores clustering.

Reporting

level(#); see [R] **Estimation options**.

display_options: noci, nopvalues, noomitted, vsquish, noemptycells, baselevels, allbaselevels, nofvlabel, fvwrap(#), fvwrapon(*style*), cformat(%*fmt*), pformat(%*fmt*), sformat(%*fmt*), and nolstretch; see [R] **Estimation options**.

Optimization

[no]log displays or suppresses a log showing the progress of the estimation. By default, one-line messages indicating when each lasso estimation begins are shown. Specify verbose to see a more detailed log.

verbose displays a verbose log showing the iterations of each lasso estimation. This option is useful when doing selection(cv) or selection(adaptive). It allows you to monitor the progress of the lasso estimations for these selection methods, which can be time consuming when there are many *othervars* specified in controls() or many *instrumvars*.

rseed(#) sets the random-number seed. This option can be used to reproduce results for selection(cv) and selection(adaptive). The default selection method selection(plugin) does not use random numbers. rseed(#) is equivalent to typing set seed # prior to running poivregress. See [R] **set seed**.

Advanced

lasso(*varlist*, *lasso_options*) lets you set different options for different lassos, or advanced options for all lassos. You specify a *varlist* followed by the options you want to apply to the lassos for these variables, where *varlist* consists of one or more variables from *depvar*, *exovars*, or *endovars*. _all or * may be used to specify *depvar* and all *exovars* and *endovars*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are selection(...), grid(...), stop(#), tolerance(#), dtolerance(#), and cvtolerance(#).

When lasso(*varlist*, selection(...)) is specified, it overrides any global selection() option for the variables in *varlist*. It also overrides the global sqrtlasso option for these variables. See [LASSO] **lasso options**.

sqrtlasso(*varlist*, *lasso_options*) works like the option lasso(), except square-root lassos for the variables in *varlist* are done rather than regular lassos. *varlist* consists of one or more variables from *depvar*, *exovars*, or *endovars*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are selection(...), grid(...), stop(#), tolerance(#), dtolerance(#), and cvtolerance(#). When sqrtlasso(*varlist*, selection(...)) is specified, it overrides any global selection() option for the variables in *varlist*. See [LASSO] **lasso options**.

The following options are available with poivregress but are not shown in the dialog box:

reestimate is an advanced option that refits the poivregress model based on changes made to the underlying lassos using lassoselect. After running poivregress, you can select a different λ^* for one or more of the lassos estimated by poivregress. After selecting λ^*, you type poivregress, reestimate to refit the poivregress model based on the newly selected λ's.

reestimate may be combined only with reporting options.

noheader prevents the coefficient table header from being displayed.

coeflegend; see [R] **Estimation options**.

Remarks and examples

poivregress performs partialing-out lasso instrumental-variables linear regression. This command estimates coefficients, standard errors, and confidence intervals and performs tests for variables of interest, both exogenous and endogenous, while using lassos to select from among potential control variables and instruments.

The instrumental-variables linear regression model is

$$y = \mathbf{d}\boldsymbol{\alpha}_d' + \mathbf{f}\boldsymbol{\alpha}_f' + \mathbf{x}\boldsymbol{\beta}' + \epsilon$$

where **d** are the endogenous variables, **f** are the exogenous variables for which we wish to make inferences, and **x** are the potential control variables from which the lassos select. In addition, lassos select from potential instrumental variables, **z**. poivregress reports estimated coefficients for $\boldsymbol{\alpha}_d$ and $\boldsymbol{\alpha}_f$. However, partialing-out does not provide estimates of the coefficients on the control variables or their standard errors. No estimation results can be reported for $\boldsymbol{\beta}$.

For an introduction to the partialing-out lasso method for inference, as well as the double-selection and cross-fit partialing-out methods, see [LASSO] **Lasso inference intro**.

Examples that demonstrate how to use poivregress and the other lasso inference commands are presented in [LASSO] **Inference examples**. In particular, we recommend reading *1 Overview* for an introduction to the examples and to the vl command, which provides tools for working with the large lists of variables that are often included when using lasso methods. See *2 Fitting and interpreting inferential models* for comparisons of the different methods of fitting inferential models that are available in Stata. See *6 Fitting an inferential model with endogenous covariates* for examples and discussion specific to models that account for endogenous covariates.

If you are interested in digging deeper into the lassos that are used to select controls, see *5 Exploring inferential model lassos* in [LASSO] **Inference examples**.

Stored results

poivregress stores the following in e():

Scalars	
e(N)	number of observations
e(N_clust)	number of clusters
e(k_varsofinterest)	number of variables of interest
e(k_controls)	number of potential control variables
e(k_controls_sel)	number of selected control variables
e(k_inst)	number of potential instruments
e(k_inst_sel)	number of selected instruments
e(df)	degrees of freedom for test of variables of interest
e(chi2)	χ^2
e(p)	p-value for test of variables of interest
e(rank)	rank of e(V)
Macros	
e(cmd)	poivregress
e(cmdline)	command as typed
e(depvar)	name of dependent variable
e(lasso_depvars)	names of dependent variables for all lassos
e(varsofinterest)	variables of interest
e(controls)	potential control variables
e(controls_sel)	selected control variables
e(exog)	exogenous variables
e(endog)	endogenous variables
e(inst)	potential instruments
e(inst_sel)	selected instruments
e(model)	linear
e(title)	title in estimation output
e(clustvar)	name of cluster variable
e(chi2type)	Wald; type of χ^2 test
e(vce)	*vcetype* specified in vce()
e(vcetype)	title used to label Std. err.
e(rngstate)	random-number state used
e(properties)	b V
e(predict)	program used to implement predict
e(select_cmd)	program used to implement lassoselect
e(marginsnotok)	predictions disallowed by margins
e(asbalanced)	factor variables fvset as asbalanced
e(asobserved)	factor variables fvset as asobserved
Matrices	
e(b)	coefficient vector
e(V)	variance–covariance matrix of the estimators
Functions	
e(sample)	marks estimation sample

In addition to the above, the following is stored in r():

Matrices	
r(table)	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in r() are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

poivregress implements partialing-out lasso instrumental-variables regression described in Chernozhukov, Hansen, and Spindler (2015). The model is

$$y = \mathbf{d}\boldsymbol{\alpha}_d' + \mathbf{f}\boldsymbol{\alpha}_f' + \mathbf{x}\boldsymbol{\beta}' + \epsilon$$

where $\mathbf{d}$ contains the J_d endogenous covariates of interest, $\mathbf{f}$ contains the J_f exogenous covariates of interest, and $\mathbf{x}$ contains the p_x controls. We also have p_z outside instrumental variables, denoted by $\mathbf{z}$, that are correlated with $\mathbf{d}$ but not with ϵ. The number of controls in $\mathbf{x}$ and the number of instruments in $\mathbf{z}$ can be large and, in theory, can grow with the sample size; however, the number of nonzero elements in $\boldsymbol{\beta}$ and nonzero coefficients of $\mathbf{z}$ must not be too large, which is to say that the model must be sparse. See *Stata commands for inference* in [LASSO] **Lasso intro** for a discussion on what it means for the model to be sparse.

In the following algorithm, each lasso can choose the lasso penalty parameter (λ^*) using the plugin estimator, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

Partialing-out lasso instrumental-variables regression algorithm

1. Perform a linear lasso of y on $\mathbf{x}$, and denote the selected controls by $\tilde{\mathbf{x}}_y$.

   Fit a linear regression of y on $\tilde{\mathbf{x}}_y$, and denote the residual for the ith observation from this regression by $\tilde{\rho}_i$.

2. For $j = 1, \dots, J_d$, perform a linear lasso of d_j on $\mathbf{f}$, $\mathbf{x}$, and $\mathbf{z}$, denote the selected controls by $\tilde{\mathbf{x}}_{d,j}$, and denote the selected instruments by $\tilde{\mathbf{z}}_j$.

   Fit a linear regression of d_j on $\mathbf{f}$, $\tilde{\mathbf{x}}_{d,j}$, and $\tilde{\mathbf{z}}_j$, and denote the linear prediction from this regression by $\hat{d}_j$.

   Perform a linear lasso of $\hat{d}_j$ on the controls $\mathbf{x}$, and denote the selected controls by $\check{\mathbf{x}}_{d,j}$.

   Fit a linear regression of $\hat{d}_j$ on $\check{\mathbf{x}}_{d,j}$, let $\check{\boldsymbol{\beta}}_j$ be the estimated coefficients, and denote the residuals from this regression by $\check{d}_j$, with its ith observation denoted by $\check{d}_{j,i}$.

   Also compute the "residuals" for the levels

   $$\tilde{d}_j = d_j - \check{\mathbf{x}}_j\check{\boldsymbol{\beta}}_j$$

   and denote its ith observation by $\tilde{d}_{j,i}$.

3. For $j = 1, \dots, J_f$, perform a linear lasso of f_j on the controls $\mathbf{x}$, and denote the selected controls by $\tilde{\mathbf{x}}_{f,j}$.

   Fit a linear regression of f_j on $\tilde{\mathbf{x}}_{f,j}$, and denote the residual for the ith observation by $\tilde{f}_{j,i}$.

4. Form the vector of instruments

   $$\mathbf{w}_i = \left(\check{d}_{1,i}, \dots, \check{d}_{J_d,i}, \tilde{f}_{1,i}, \dots, \tilde{f}_{J_f,i}\right)$$

5. Form the vector of partialed-out covariates

   $$\mathbf{p}_i = \left(\tilde{d}_{1,i}, \dots, \tilde{d}_{J_d,i}, \tilde{f}_{1,i}, \dots, \tilde{f}_{J_f,i}\right)$$

6. Compute $\widehat{\boldsymbol{\alpha}}$ by solving the following $J_d + J_f$ sample-moment equations.

$$\frac{1}{n}\sum_{i=1}^{n} \mathbf{w}_i'(\tilde{\rho}_i - \mathbf{p}_i\widehat{\boldsymbol{\alpha}}') = \mathbf{0}$$

7. The variance for $\widehat{\boldsymbol{\alpha}}$ is estimated by

$$\widehat{\mathbf{Var}}(\widehat{\boldsymbol{\alpha}}) = \frac{1}{n}\left(\hat{\mathbf{J}}_0^{-1}\right)\widehat{\boldsymbol{\Psi}}\left(\hat{\mathbf{J}}_0^{-1}\right)'$$

where

$$\hat{\mathbf{J}}_0 = \frac{1}{n}\sum_{i=1}^{n} \mathbf{w}_i'\mathbf{p}_i$$

$$\widehat{\boldsymbol{\Psi}} = \frac{1}{n}\sum_{i=1}^{n} \widehat{\boldsymbol{\psi}}_i\widehat{\boldsymbol{\psi}}_i'$$

$$\widehat{\boldsymbol{\psi}}_i = \mathbf{w}_i'(\tilde{\rho}_i - \mathbf{p}_i\widehat{\boldsymbol{\alpha}}')$$

See *Methods and formulas* in [LASSO] **lasso** for details on how the lassos in steps 1, 2, and 3 choose their penalty parameters (λ^*).

Reference

Chernozhukov, V., C. B. Hansen, and M. Spindler. 2015. Valid post-selection and post-regularization inference: An elementary, general approach. *Annual Review of Economics* 7: 649–688. https://doi.org/10.1146/annurev-economics-012315-015826.

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **xpoivregress** — Cross-fit partialing-out lasso instrumental-variables regression

[R] **ivregress** — Single-equation instrumental-variables regression

[U] 20 Estimation and postestimation commands

pologit — Partialing-out lasso logistic regression

Description	Quick start	Menu	Syntax	Options
Remarks and examples	Stored results	Methods and formulas	Reference	Also see

Description

`pologit` fits a lasso logistic regression model and reports odds ratios along with standard errors, test statistics, and confidence intervals for specified covariates of interest. The partialing-out method is used to estimate effects for these variables and to select from potential control variables to be included in the model.

Quick start

Report an odds ratio from a logistic regression of `y` on `d1`, and include `x1` to `x100` as potential control variables to be selected by lassos

```
pologit y d1, controls(x1-x100)
```

Same as above, and estimate odds ratios for the levels of categorical `d2`

```
pologit y d1 i.d2, controls(x1-x100)
```

Use cross-validation (CV) instead of a plugin iterative formula to select the optimal λ^* in each lasso

```
pologit y d1 i.d2, controls(x1-x100) selection(cv)
```

Same as above, and set a random-number seed for reproducibility

```
pologit y d1 i.d2, controls(x1-x100) selection(cv) rseed(28)
```

Specify CV for the lasso for `y` only, with the stopping rule criterion turned off

```
pologit y d1 i.d2, controls(x1-x100) lasso(y, selection(cv), stop(0))
```

Same as above, but apply the option to the lassos for `y`, `d1`, and `i.d2`

```
pologit y d1 i.d2, controls(x1-x100) lasso(*, selection(cv), stop(0))
```

Compute lassos beyond the CV minimum to get full coefficient paths, knots, etc.

```
pologit y d1 i.d2, controls(x1-x100) lasso(*, selection(cv, alllambdas))
```

Menu

Statistics > Lasso > Lasso inferential models > Binary outcomes > Partialing-out logit model

Syntax

```
pologit depvar varsofinterest [if] [in] ,
    controls([(alwaysvars)] othervars) [options]
```

varsofinterest are variables for which coefficients and their standard errors are estimated.

options	Description
Model	
* `controls([(`*alwaysvars*`)]` *othervars*`)`	*alwaysvars* and *othervars* make up the set of control variables; *alwaysvars* are always included; lassos choose whether to include or exclude *othervars*
`selection(plugin)`	use a plugin iterative formula to select an optimal value of the lasso penalty parameter λ^* for each lasso; the default
`selection(cv)`	use CV to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(adaptive)`	use adaptive lasso to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(bic)`	use BIC to select an optimal value of the lasso penalty parameter λ^* for each lasso
`sqrtlasso`	use square-root lassos for *varsofinterest*
`missingok`	after fitting lassos, ignore missing values in any *othervars* not selected, and include these observations in the final model
`offset(`*varname*`)`	include *varname* in the lasso and model for *depvar* with its coefficient constrained to be 1
SE/Robust	
`vce(`*vcetype*`)`	*vcetype* may be `robust` (the default) or `cluster` *clustvar*
Reporting	
`level(#)`	set confidence level; default is `level(95)`
`or`	report odds ratios; the default
`coef`	report estimated coefficients
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
Optimization	
`[no]log`	display or suppress an iteration log
`verbose`	display a verbose iteration log
`rseed(#)`	set random-number seed
Advanced	
`lasso(`*varlist*`,` *lasso_options*`)`	specify options for the lassos for variables in *varlist*; may be repeated
`sqrtlasso(`*varlist*`,` *lasso_options*`)`	specify options for square-root lassos for variables in *varlist*; may be repeated

`reestimate`	refit the model after using `lassoselect` to select a different λ^*
`noheader`	do not display the header on the coefficient table
`coeflegend`	display legend instead of statistics

*`controls()` is required.

varsofinterest, *alwaysvars*, and *othervars* may contain factor variables. Base levels of factor variables cannot be set for *alwaysvars* and *othervars*. See **[U] 11.4.3 Factor variables**.

`collect` is allowed; see **[U] 11.1.10 Prefix commands**.

`reestimate`, `noheader`, and `coeflegend` do not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

`controls(`[`(`*alwaysvars*`)`] *othervars*`)` specifies the set of control variables, which control for omitted variables. Control variables are also known as confounding variables. `pologit` fits lassos for *depvar* and each of the *varsofinterest*. *alwaysvars* are variables that are always to be included in these lassos. *alwaysvars* are optional. *othervars* are variables that each lasso will choose to include or exclude. That is, each lasso will select a subset of *othervars*. The selected subset of *othervars* may differ across lassos. `controls()` is required.

`selection(plugin | cv | adaptive | bic)` specifies the selection method for choosing an optimal value of the lasso penalty parameter λ^* for each lasso or square-root lasso estimation. Separate lassos are estimated for *depvar* and each variable in *varsofinterest*. Specifying `selection()` changes the selection method for all of these lassos. You can specify different selection methods for different lassos using the option `lasso()` or `sqrtlasso()`. When `lasso()` or `sqrtlasso()` is used to specify a different selection method for the lassos of some variables, they override the global setting made using `selection()` for the specified variables.

`selection(plugin)` is the default. It selects λ^* based on a "plugin" iterative formula dependent on the data. See [LASSO] **lasso options**.

`selection(cv)` selects the λ^* that gives the minimum of the CV function. See [LASSO] **lasso options**.

`selection(adaptive)` selects λ^* using the adaptive lasso selection method. It cannot be specified when `sqrtlasso` is specified. See [LASSO] **lasso options**.

`selection(bic)` selects the λ^* that gives the minimum of the BIC function. See [LASSO] **lasso options**.

`sqrtlasso` specifies that square-root lassos be done rather than regular lassos for the *varsofinterest*. This option does not apply to *depvar*. Square-root lassos are linear models, and the lasso for *depvar* is always a logit lasso. The option `lasso()` can be used with `sqrtlasso` to specify that regular lasso be done for some variables, overriding the global `sqrtlasso` setting for these variables. See [LASSO] **lasso options**.

`missingok` specifies that, after fitting lassos, the estimation sample be redefined based on only the nonmissing observations of variables in the final model. In all cases, any observation with missing values for *depvar*, *varsofinterest*, *alwaysvars*, and *othervars* is omitted from the estimation sample for the lassos. By default, the same sample is used for calculation of the coefficients of the *varsofinterest* and their standard errors.

When missingok is specified, the initial estimation sample is the same as the default, but the sample used for the calculation of the coefficients of the *varsofinterest* can be larger. Now observations with missing values for any *othervars* not selected will be added to the estimation sample (provided there are no missing values for any of the variables in the final model).

missingok may produce more efficient estimates when data are missing completely at random. It does, however, have the consequence that estimation samples can change when selected variables differ in models fit using different selection methods. That is, when *othervars* contain missing values, the estimation sample for a model fit using the default selection(plugin) will likely differ from the estimation sample for a model fit using, for example, selection(cv).

offset(*varname*) specifies that *varname* be included in the lasso and model for *depvar* with its coefficient constrained to be 1.

SE/Robust

vce(*vcetype*) specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (robust) and that allow for intragroup correlation (cluster *clustvar*); see [R] ***vce_option***.

When vce(cluster *clustvar*) is specified, all lassos also account for clustering. For each lasso, this affects how the log-likelihood function is computed and how the sample is split in cross-validation; see *Methods and formulas* in [LASSO] **lasso**. Specifying vce(cluster *clustvar*) may lead to different selected controls and therefore to different point estimates for your variable of interest when compared to the estimation that ignores clustering.

Reporting

level(#); see [R] **Estimation options**.

or reports the estimated coefficients transformed to odds ratios, that is, e^{α}. Standard errors and confidence intervals are similarly transformed. or is the default.

coef reports the estimated coefficients α rather than the odds ratios (e^{α}). This option affects how results are displayed, not how they are estimated. coef may be specified at estimation or when replaying previously estimated results.

display_options: noci, nopvalues, noomitted, vsquish, noemptycells, baselevels, allbaselevels, nofvlabel, fvwrap(#), fvwrapon(*style*), cformat(%*fmt*), pformat(%*fmt*), sformat(%*fmt*), and nolstretch; see [R] **Estimation options**.

Optimization

[no]log displays or suppresses a log showing the progress of the estimation. By default, one-line messages indicating when each lasso estimation begins are shown. Specify verbose to see a more detailed log.

verbose displays a verbose log showing the iterations of each lasso estimation. This option is useful when doing selection(cv) or selection(adaptive). It allows you to monitor the progress of the lasso estimations for these selection methods, which can be time consuming when there are many *othervars* specified in controls().

rseed(#) sets the random-number seed. This option can be used to reproduce results for selection(cv) and selection(adaptive). The default selection method selection(plugin) does not use random numbers. rseed(#) is equivalent to typing set seed # prior to running pologit. See [R] **set seed**.

Advanced

`lasso(`*varlist*`,` *lasso_options*`)` lets you set different options for different lassos, or advanced options for all lassos. You specify a *varlist* followed by the options you want to apply to the lassos for these variables. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. `_all` or `*` may be used to specify *depvar* and all *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `lasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. It also overrides the global `sqrtlasso` option for these variables. See [LASSO] **lasso options**.

`sqrtlasso(`*varlist*`,` *lasso_options*`)` works like the option `lasso()`, except square-root lassos for the variables in *varlist* are done rather than regular lassos. *varlist* consists of one or more variables from *varsofinterest*. Square-root lassos are linear models, and this option cannot be used with *depvar*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `sqrtlasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. See [LASSO] **lasso options**.

The following options are available with `pologit` but are not shown in the dialog box:

`reestimate` is an advanced option that refits the `pologit` model based on changes made to the underlying lassos using `lassoselect`. After running `pologit`, you can select a different λ^* for one or more of the lassos estimated by `pologit`. After selecting λ^*, you type `pologit, reestimate` to refit the `pologit` model based on the newly selected λ's.

`reestimate` may be combined only with reporting options.

`noheader` prevents the coefficient table header from being displayed.

`coeflegend`; see [R] **Estimation options**.

Remarks and examples

`pologit` performs partialing-out lasso logistic regression. This command estimates odds ratios, standard errors, and confidence intervals and performs tests for variables of interest while using lassos to select from among potential control variables.

The logistic regression model is

$$\Pr(y = 1|\mathbf{d}, \mathbf{x}) = \frac{\exp(\mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}')}{1 + \exp(\mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}')}$$

where **d** are the variables for which we wish to make inferences and **x** are the potential control variables from which the lassos select. `pologit` estimates the $\boldsymbol{\alpha}$ coefficients and reports the corresponding odds ratios, e^{α}. However, partialing-out does not provide estimates of the coefficients on the control variables ($\boldsymbol{\beta}$) or their standard errors. No estimation results can be reported for $\boldsymbol{\beta}$.

For an introduction to the partialing-out lasso method for inference, as well as the double-selection and cross-fit partialing-out methods, see [LASSO] **Lasso inference intro**.

Examples that demonstrate how to use `pologit` and the other lasso inference commands are presented in [LASSO] **Inference examples**. In particular, we recommend reading *1 Overview* for an introduction to the examples and to the `vl` command, which provides tools for working with the large lists of variables that are often included when using lasso methods. See *2 Fitting and interpreting inferential models* for comparisons of the different methods of fitting inferential models that are available in Stata. Everything we say there about methods of selection is applicable to both linear and nonlinear models. See *3 Fitting logit inferential models to binary outcomes. What is different?* for examples and discussion specific to logistic regression models. The primary difference from linear models involves interpreting the results.

If you are interested in digging deeper into the lassos that are used to select controls, see *5 Exploring inferential model lassos* in [LASSO] **Inference examples**.

Stored results

pologit stores the following in e():

Scalars	
e(N)	number of observations
e(N_clust)	number of clusters
e(k_varsofinterest)	number of variables of interest
e(k_controls)	number of potential control variables
e(k_controls_sel)	number of selected control variables
e(df)	degrees of freedom for test of variables of interest
e(chi2)	χ^2
e(p)	p-value for test of variables of interest
e(rank)	rank of e(V)
Macros	
e(cmd)	pologit
e(cmdline)	command as typed
e(depvar)	name of dependent variable
e(lasso_depvars)	names of dependent variables for all lassos
e(varsofinterest)	variables of interest
e(controls)	potential control variables
e(controls_sel)	selected control variables
e(model)	logit
e(title)	title in estimation output
e(offset)	linear offset variable
e(clustvar)	name of cluster variable
e(chi2type)	Wald; type of χ^2 test
e(vce)	*vcetype* specified in vce()
e(vcetype)	title used to label Std. err.
e(rngstate)	random-number state used
e(properties)	b V
e(predict)	program used to implement predict
e(select_cmd)	program used to implement lassoselect
e(marginsnotok)	predictions disallowed by margins
e(asbalanced)	factor variables fvset as asbalanced
e(asobserved)	factor variables fvset as asobserved
Matrices	
e(b)	coefficient vector
e(V)	variance–covariance matrix of the estimators
Functions	
e(sample)	marks estimation sample

In addition to the above, the following is stored in r():

Matrices	
r(table)	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

`pologit` implements partialing-out lasso logit regression (POLLR) as described in Belloni, Chernozhukov, and Wei (2016, table 1 and sec. 2.1). The regression model is

$$\mathbf{E}[y|\mathbf{d},\,\mathbf{x}] = G(\mathbf{d}\boldsymbol{\alpha}' + \beta_0 + \mathbf{x}\boldsymbol{\beta}')$$

where $G(a) = \exp(a)/\{1 + \exp(a)\}$, $\mathbf{d}$ contains the J covariates of interest, and $\mathbf{x}$ contains the p controls. The number of covariates in $\mathbf{d}$ must be small and fixed. The number of controls in $\mathbf{x}$ can be large and, in theory, can grow with the sample size; however, the number of nonzero elements in $\boldsymbol{\beta}$ must not be too large, which is to say that the model must be sparse.

POLLR algorithm

1. Perform a logit lasso of y on $\mathbf{d}$ and $\mathbf{x}$, and denote the selected controls by $\tilde{\mathbf{x}}$.

   This logit lasso can choose the lasso penalty parameter (λ^*) using the plugin estimator, adaptive lasso, or CV. The plugin value is the default.

2. Fit a logit regression of y on $\mathbf{d}$ and $\tilde{\mathbf{x}}$, denoting the estimated coefficient vectors by $\widetilde{\boldsymbol{\alpha}}$ and $\widetilde{\boldsymbol{\beta}}$, respectively.

3. Let $\tilde{s}_i = \tilde{\mathbf{x}}_i\widetilde{\boldsymbol{\beta}}'$ be the ith observation of the predicted value of $\mathbf{x}\boldsymbol{\beta}'$ and $w_i = G'(\mathbf{d}_i\widetilde{\boldsymbol{\alpha}}' + \tilde{s}_i)$ be the ith observation of the predicted value of the derivative of $G(\cdot)$.

4. For $j = 1, \ldots, J$, perform a linear lasso of d_j on $\mathbf{x}$ using observation-level weights w_i, and denote the selected controls by $\check{\mathbf{x}}_j$.

   Each of these lassos can choose the lasso penalty parameter (λ_j^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

5. For $j = 1, \ldots, J$, fit a linear regression of d_j on the selected controls $\check{\mathbf{x}}_j$ using observation-level weights w_i, denote the unweighted residuals by $\tilde{d}_j$, with $\tilde{d}_{j,i}$ its ith observation, and create the instrument z_j with ith observation given by $z_{j,i} = \tilde{d}_{j,i}$.

   Collect the J instruments for the ith observation into the vector $\mathbf{z}_i = (z_{1,i}, \ldots, z_{J,i})$.

6. Compute $\widehat{\boldsymbol{\alpha}}$ by solving the following J sample-moment equations

$$\frac{1}{n}\sum_{i=1}^{n}\{y_i - G(\mathbf{d}_i\boldsymbol{\alpha}' + \tilde{s}_i)\}\,\mathbf{z}_i' = \mathbf{0}$$

7. Store the point estimates $\widehat{\boldsymbol{\alpha}}$ in `e(b)` and their variance estimates (VCE) in `e(V)`.

   The VCE is estimated by the robust estimator for method of moments.

See *Methods and formulas* in [LASSO] **lasso** for details on how the lassos in steps 1 and 4 choose their penalty parameter (λ^*).

Reference

Belloni, A., V. Chernozhukov, and Y. Wei. 2016. Post-selection inference for generalized linear models with many controls. *Journal of Business and Economic Statistics* 34: 606–619. https://doi.org/10.1080/07350015.2016.1166116.

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **dslogit** — Double-selection lasso logistic regression

[LASSO] **xpologit** — Cross-fit partialing-out lasso logistic regression

[R] **logit** — Logistic regression, reporting coefficients

[R] **logistic** — Logistic regression, reporting odds ratios

[U] 20 Estimation and postestimation commands

popoisson — Partialing-out lasso Poisson regression

Description

popoisson fits a lasso Poisson regression model and reports incidence-rate ratios along with standard errors, test statistics, and confidence intervals for specified covariates of interest. The partialing-out method is used to estimate effects for these variables and to select from potential control variables to be included in the model.

Quick start

Report an incidence-rate ratio from a Poisson regression of y on d1, and include x1 to x100 as potential control variables to be selected by lassos

```
popoisson y d1, controls(x1-x100)
```

Same as above, and estimate incidence-rate ratios for the levels of categorical d2

```
popoisson y d1 i.d2, controls(x1-x100)
```

Use cross-validation (CV) instead of a plugin iterative formula to select the optimal λ^* in each lasso

```
popoisson y d1 i.d2, controls(x1-x100) selection(cv)
```

Same as above, and set a random-number seed for reproducibility

```
popoisson y d1 i.d2, controls(x1-x100) selection(cv) rseed(28)
```

Specify CV for the lasso for y only, with the stopping rule criterion turned off

```
popoisson y d1 i.d2, controls(x1-x100) lasso(y, selection(cv), stop(0))
```

Same as above, but apply the option to the lassos for y, d1, and i.d2

```
popoisson y d1 i.d2, controls(x1-x100) lasso(*, selection(cv), stop(0))
```

Compute lassos beyond the CV minimum to get full coefficient paths, knots, etc.

```
popoisson y d1 i.d2, controls(x1-x100) ///
    lasso(*, selection(cv, alllambdas))
```

Menu

Statistics > Lasso > Lasso inferential models > Count outcomes > Partialing-out Poisson model

Syntax

popoisson *depvar varsofinterest* [*if*] [*in*] ,

controls([(*alwaysvars*)] *othervars*) [*options*]

varsofinterest are variables for which coefficients and their standard errors are estimated.

options	Description
Model	
* controls([(*alwaysvars*)] *othervars*)	*alwaysvars* and *othervars* make up the set of control variables; *alwaysvars* are always included; lassos choose whether to include or exclude *othervars*
selection(plugin)	use a plugin iterative formula to select an optimal value of the lasso penalty parameter λ^* for each lasso; the default
selection(cv)	use CV to select an optimal value of the lasso penalty parameter λ^* for each lasso
selection(adaptive)	use adaptive lasso to select an optimal value of the lasso penalty parameter λ^* for each lasso
selection(bic)	use BIC to select an optimal value of the lasso penalty parameter λ^* for each lasso
sqrtlasso	use square-root lassos for *varsofinterest*
missingok	after fitting lassos, ignore missing values in any *othervars* not selected, and include these observations in the final model
offset(*varname*$_o$)	include *varname*$_o$ in the lasso and model for *depvar* with its coefficient constrained to be 1
exposure(*varname*$_e$)	include ln(*varname*$_e$) in the lasso and model for *depvar* with its coefficient constrained to be 1
SE/Robust	
vce(*vcetype*)	*vcetype* may be robust (the default) or cluster *clustvar*
Reporting	
level(#)	set confidence level; default is level(95)
irr	report incidence-rate ratios; the default
coef	report estimated coefficients
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
Optimization	
[no]log	display or suppress an iteration log
verbose	display a verbose iteration log
rseed(#)	set random-number seed
Advanced	
lasso(*varlist*, *lasso_options*)	specify options for the lassos for variables in *varlist*; may be repeated
sqrtlasso(*varlist*, *lasso_options*)	specify options for square-root lassos for variables in *varlist*; may be repeated

`reestimate`	refit the model after using `lassoselect` to select a different λ^*
`noheader`	do not display the header on the coefficient table
`coeflegend`	display legend instead of statistics

*`controls()` is required.

varsofinterest, *alwaysvars*, and *othervars* may contain factor variables. Base levels of factor variables cannot be set for *alwaysvars* and *othervars*. See **[U] 11.4.3 Factor variables**.

`collect` is allowed; see **[U] 11.1.10 Prefix commands**.

`reestimate`, `noheader`, and `coeflegend` do not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

`controls(`[`(`*alwaysvars*`)`] *othervars*`)` specifies the set of control variables, which control for omitted variables. Control variables are also known as confounding variables. `popoisson` fits lassos for *depvar* and each of the *varsofinterest*. *alwaysvars* are variables that are always to be included in these lassos. *alwaysvars* are optional. *othervars* are variables that each lasso will choose to include or exclude. That is, each lasso will select a subset of *othervars*. The selected subset of *othervars* may differ across lassos. `controls()` is required.

`selection(plugin | cv | adaptive | bic)` specifies the selection method for choosing an optimal value of the lasso penalty parameter λ^* for each lasso or square-root lasso estimation. Separate lassos are estimated for *depvar* and each variable in *varsofinterest*. Specifying `selection()` changes the selection method for all of these lassos. You can specify different selection methods for different lassos using the option `lasso()` or `sqrtlasso()`. When `lasso()` or `sqrtlasso()` is used to specify a different selection method for the lassos of some variables, they override the global setting made using `selection()` for the specified variables.

`selection(plugin)` is the default. It selects λ^* based on a "plugin" iterative formula dependent on the data. See [LASSO] **lasso options**.

`selection(cv)` selects the λ^* that gives the minimum of the CV function. See [LASSO] **lasso options**.

`selection(adaptive)` selects λ^* using the adaptive lasso selection method. It cannot be specified when `sqrtlasso` is specified. See [LASSO] **lasso options**.

`selection(bic)` selects the λ^* that gives the minimum of the BIC function. See [LASSO] **lasso options**.

`sqrtlasso` specifies that square-root lassos be done rather than regular lassos for the *varsofinterest*. This option does not apply to *depvar*. Square-root lassos are linear models, and the lasso for *depvar* is always a Poisson lasso. The option `lasso()` can be used with `sqrtlasso` to specify that regular lasso be done for some variables, overriding the global `sqrtlasso` setting for these variables. See [LASSO] **lasso options**.

`missingok` specifies that, after fitting lassos, the estimation sample be redefined based on only the nonmissing observations of variables in the final model. In all cases, any observation with missing values for *depvar*, *varsofinterest*, *alwaysvars*, and *othervars* is omitted from the estimation sample for the lassos. By default, the same sample is used for calculation of the coefficients of the *varsofinterest* and their standard errors.

When `missingok` is specified, the initial estimation sample is the same as the default, but the sample used for the calculation of the coefficients of the *varsofinterest* can be larger. Now observations with missing values for any *othervars* not selected will be added to the estimation sample (provided there are no missing values for any of the variables in the final model).

`missingok` may produce more efficient estimates when data are missing completely at random. It does, however, have the consequence that estimation samples can change when selected variables differ in models fit using different selection methods. That is, when *othervars* contain missing values, the estimation sample for a model fit using the default `selection(plugin)` will likely differ from the estimation sample for a model fit using, for example, `selection(cv)`.

`offset(`*varname*$_o$`)` specifies that *varname*$_o$ be included in the lasso and model for *depvar* with its coefficient constrained to be 1.

`exposure(`*varname*$_e$`)` specifies that ln(*varname*$_e$) be included in the lasso and model for *depvar* with its coefficient constrained to be 1.

SE/Robust

`vce(`*vcetype*`)` specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (`robust`) and that allow for intragroup correlation (`cluster` *clustvar*); see [R] ***vce_option***.

When `vce(cluster` *clustvar*`)` is specified, all lassos also account for clustering. For each lasso, this affects how the log-likelihood function is computed and how the sample is split in cross-validation; see *Methods and formulas* in [LASSO] **lasso**. Specifying `vce(cluster` *clustvar*`)` may lead to different selected controls and therefore to different point estimates for your variable of interest when compared to the estimation that ignores clustering.

Reporting

`level(#)`; see [R] **Estimation options**.

`irr` reports estimated coefficients transformed to incidence-rate ratios, that is, e^{α}. Standard errors and confidence intervals are similarly transformed. `irr` is the default.

`coef` reports the estimated coefficients α rather than the incidence-rate ratios, e^{α}. This option affects how results are displayed, not how they are estimated. `coef` may be specified at estimation or when replaying previously estimated results.

display_options: `noci`, `nopvalues`, `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `fvwrap(#)`, `fvwrapon(`*style*`)`, `cformat(`*%fmt*`)`, `pformat(`*%fmt*`)`, `sformat(`*%fmt*`)`, and `nolstretch`; see [R] **Estimation options**.

Optimization

`[no]log` displays or suppresses a log showing the progress of the estimation. By default, one-line messages indicating when each lasso estimation begins are shown. Specify `verbose` to see a more detailed log.

`verbose` displays a verbose log showing the iterations of each lasso estimation. This option is useful when doing `selection(cv)` or `selection(adaptive)`. It allows you to monitor the progress of the lasso estimations for these selection methods, which can be time consuming when there are many *othervars* specified in `controls()`.

`rseed(#)` sets the random-number seed. This option can be used to reproduce results for `selection(cv)` and `selection(adaptive)`. The default selection method `selection(plugin)` does not use random numbers. `rseed(#)` is equivalent to typing `set seed #` prior to running `popoisson`. See [R] **set seed**.

Advanced

`lasso(`*varlist*`,` *lasso_options*`)` lets you set different options for different lassos, or advanced options for all lassos. You specify a *varlist* followed by the options you want to apply to the lassos for these variables. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. `_all` or `*` may be used to specify *depvar* and all *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `lasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. It also overrides the global `sqrtlasso` option for these variables. See [LASSO] **lasso options**.

`sqrtlasso(`*varlist*`,` *lasso_options*`)` works like the option `lasso()`, except square-root lassos for the variables in *varlist* are done rather than regular lassos. *varlist* consists of one or more variables from *varsofinterest*. Square-root lassos are linear models, and this option cannot be used with *depvar*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `sqrtlasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. See [LASSO] **lasso options**.

The following options are available with `popoisson` but are not shown in the dialog box:

`reestimate` is an advanced option that refits the `popoisson` model based on changes made to the underlying lassos using `lassoselect`. After running `popoisson`, you can select a different λ^* for one or more of the lassos estimated by `popoisson`. After selecting λ^*, you type `popoisson, reestimate` to refit the `popoisson` model based on the newly selected λ^*'s.

`reestimate` may be combined only with reporting options.

`noheader` prevents the coefficient table header from being displayed.

`coeflegend`; see [R] **Estimation options**.

Remarks and examples

`popoisson` performs partialing-out lasso Poisson regression. This command estimates incidence-rate ratios, standard errors, and confidence intervals and performs tests for variables of interest while using lassos to select from among potential control variables.

The Poisson regression model is

$$\mathbf{E}[y|\mathbf{d},\, \mathbf{x}] = \exp(\mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}')$$

where **d** are the variables for which we wish to make inferences and **x** are the potential control variables from which the lassos select. `popoisson` estimates the $\boldsymbol{\alpha}$ coefficients and reports the corresponding incidence-rate ratios, e^{α}. However, partialing-out does not provide estimates of the coefficients on the control variables ($\boldsymbol{\beta}$) or their standard errors. No estimation results can be reported for $\boldsymbol{\beta}$.

For an introduction to the partialing-out lasso method for inference, as well as the double-selection and cross-fit partialing-out methods, see [LASSO] **Lasso inference intro**.

Examples that demonstrate how to use popoisson and the other lasso inference commands are presented in [LASSO] **Inference examples**. In particular, we recommend reading *1 Overview* for an introduction to the examples and to the vl command, which provides tools for working with the large lists of variables that are often included when using lasso methods. See *2 Fitting and interpreting inferential models* for comparisons of the different methods of fitting inferential models that are available in Stata. Everything we say there about methods of selection is applicable to both linear and nonlinear models. See *4 Fitting inferential models to count outcomes. What is different?* for examples and discussion specific to Poisson regression models. The primary difference from linear models involves interpreting the results.

If you are interested in digging deeper into the lassos that are used to select controls, see *5 Exploring inferential model lassos* in [LASSO] **Inference examples**.

Stored results

popoisson stores the following in e():

Scalars	
e(N)	number of observations
e(N_clust)	number of clusters
e(k_varsofinterest)	number of variables of interest
e(k_controls)	number of potential control variables
e(k_controls_sel)	number of selected control variables
e(df)	degrees of freedom for test of variables of interest
e(chi2)	χ^2
e(p)	p-value for test of variables of interest
e(rank)	rank of e(V)
Macros	
e(cmd)	popoisson
e(cmdline)	command as typed
e(depvar)	name of dependent variable
e(lasso_depvars)	names of dependent variables for all lassos
e(varsofinterest)	variables of interest
e(controls)	potential control variables
e(controls_sel)	selected control variables
e(model)	poisson
e(title)	title in estimation output
e(offset)	linear offset variable
e(clustvar)	name of cluster variable
e(chi2type)	Wald; type of χ^2 test
e(vce)	*vcetype* specified in vce()
e(vcetype)	title used to label Std. err.
e(rngstate)	random-number state used
e(properties)	b V
e(predict)	program used to implement predict
e(select_cmd)	program used to implement lassoselect
e(marginsnotok)	predictions disallowed by margins
e(asbalanced)	factor variables fvset as asbalanced
e(asobserved)	factor variables fvset as asobserved
Matrices	
e(b)	coefficient vector
e(V)	variance–covariance matrix of the estimators
Functions	
e(sample)	marks estimation sample

In addition to the above, the following is stored in `r()`:

Matrices

`r(table)`	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

`popoisson` implements partialing-out lasso Poisson regression (POLPR) as described in Belloni, Chernozhukov, and Wei (2016, table 1 and sec. 2.1). The regression model is

$$\mathbf{E}[y|\mathbf{d},\, \mathbf{x}] = G(\mathbf{d}\boldsymbol{\alpha}' + \beta_0 + \mathbf{x}\boldsymbol{\beta}')$$

where $G(a) = \exp(a)$, $\mathbf{d}$ contains the J covariates of interest, and $\mathbf{x}$ contains the p controls. The number of covariates in $\mathbf{d}$ must be small and fixed. The number of controls in $\mathbf{x}$ can be large and, in theory, can grow with the sample size; however, the number of nonzero elements in $\boldsymbol{\beta}$ must not be too large, which is to say that the model must be sparse.

POLPR algorithm

1. Perform a Poisson lasso of y on $\mathbf{d}$ and $\mathbf{x}$, and denote the selected controls by $\tilde{\mathbf{x}}$.

   This Poisson lasso can choose the lasso penalty parameter (λ^*) using the plugin estimator, adaptive lasso, or CV. The plugin value is the default.

2. Fit a Poisson regression of y on $\mathbf{d}$ and $\tilde{\mathbf{x}}$, denoting the estimated coefficient vectors by $\widetilde{\boldsymbol{\alpha}}$ and $\widetilde{\boldsymbol{\beta}}$, respectively.

3. Let $\tilde{s}_i = \tilde{\mathbf{x}}_i\widetilde{\boldsymbol{\beta}}'$ be the ith observation of the predicted value of $\mathbf{x}\boldsymbol{\beta}'$ and $w_i = G'(\mathbf{d}_i\widetilde{\boldsymbol{\alpha}}' + \tilde{s}_i)$ be the ith observation of the predicted value of the derivative of $G(\cdot)$.

4. For $j = 1, \ldots, J$, perform a linear lasso of d_j on $\mathbf{x}$ using observation-level weights w_i, and denote the selected controls by $\check{\mathbf{x}}_j$.

   Each of these lassos can choose the lasso penalty parameter (λ_j^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

5. For $j = 1, \ldots, J$, fit a linear regression of d_j on the selected controls $\check{\mathbf{x}}_j$ using observation-level weights w_i, denote the unweighted residuals by $\tilde{d}_j$, with $\tilde{d}_{j,i}$ its ith observation, and create the instrument z_j with ith observation given by $z_{j,i} = \tilde{d}_{j,i}$.

   Collect the J instruments for the ith observation into the vector $\mathbf{z}_i = (z_{1,i}, \ldots, z_{J,i})$.

6. Compute $\widehat{\boldsymbol{\alpha}}$ by solving the following J sample-moment equations

$$\frac{1}{n}\sum_{i=1}^{n} \{y_i - G(\mathbf{d}_i\boldsymbol{\alpha}' + \tilde{s}_i)\}\, \mathbf{z}_i' = \mathbf{0}$$

7. Store the point estimates $\widehat{\boldsymbol{\alpha}}$ in `e(b)` and their variance estimates (VCE) in `e(V)`.

   The VCE is estimated by the robust estimator for method of moments.

See *Methods and formulas* in [LASSO] **lasso** for details on how the lassos in steps 1 and 4 choose their penalty parameter (λ^*).

Reference

Belloni, A., V. Chernozhukov, and Y. Wei. 2016. Post-selection inference for generalized linear models with many controls. *Journal of Business and Economic Statistics* 34: 606–619. https://doi.org/10.1080/07350015.2016.1166116.

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **dspoisson** — Double-selection lasso Poisson regression

[LASSO] **xpopoisson** — Cross-fit partialing-out lasso Poisson regression

[R] **poisson** — Poisson regression

[U] 20 Estimation and postestimation commands

poregress — Partialing-out lasso linear regression

Description

`poregress` fits a lasso linear regression model and reports coefficients along with standard errors, test statistics, and confidence intervals for specified covariates of interest. The partialing-out method is used to estimate effects for these variables and to select from potential control variables to be included in the model.

Quick start

Estimate a coefficient for `d1` in a linear regression of `y` on `d1`, and include `x1` to `x100` as potential control variables to be selected by lassos

```
poregress y d1, controls(x1-x100)
```

Same as above, and estimate coefficients for the levels of categorical `d2`

```
poregress y d1 i.d2, controls(x1-x100)
```

Use cross-validation (CV) instead of a plugin iterative formula to select the optimal λ^* in each lasso

```
poregress y d1 i.d2, controls(x1-x100) selection(cv)
```

Same as above, and set a random-number seed for reproducibility

```
poregress y d1 i.d2, controls(x1-x100) selection(cv) rseed(28)
```

Specify CV for the lasso for `y` only, with the stopping rule criterion turned off

```
poregress y d1 i.d2, controls(x1-x100) lasso(y, selection(cv), stop(0))
```

Same as above, but apply the option to the lassos for `y`, `d1`, and `i.d2`

```
poregress y d1 i.d2, controls(x1-x100) lasso(*, selection(cv), stop(0))
```

Compute lassos beyond the CV minimum to get full coefficient paths, knots, etc.

```
poregress y d1 i.d2, controls(x1-x100) ///
   lasso(*, selection(cv, alllambdas))
```

Menu

Statistics > Lasso > Lasso inferential models > Continuous outcomes > Partialing-out model

Syntax

```
poregress depvar varsofinterest [if] [in] ,
    controls([(alwaysvars)] othervars) [options]
```

varsofinterest are variables for which coefficients and their standard errors are estimated.

options	Description
Model	
* `controls([(`*alwaysvars*`)]` *othervars*`)`	*alwaysvars* and *othervars* make up the set of control variables; *alwaysvars* are always included; lassos choose whether to include or exclude *othervars*
`selection(plugin)`	use a plugin iterative formula to select an optimal value of the lasso penalty parameter λ^* for each lasso; the default
`selection(cv)`	use CV to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(adaptive)`	use adaptive lasso to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(bic)`	use BIC to select an optimal value of the lasso penalty parameter λ^* for each lasso
`sqrtlasso`	use square-root lassos
`semi`	use semipartialing-out lasso regression estimator
`missingok`	after fitting lassos, ignore missing values in any *othervars* not selected, and include these observations in the final model
SE/Robust	
`vce(`*vcetype*`)`	*vcetype* may be `robust` (the default) or `cluster` *clustvar*
Reporting	
`level(#)`	set confidence level; default is `level(95)`
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
Optimization	
`[no]log`	display or suppress an iteration log
`verbose`	display a verbose iteration log
`rseed(#)`	set random-number seed
Advanced	
`lasso(`*varlist*`,` *lasso_options*`)`	specify options for the lassos for variables in *varlist*; may be repeated
`sqrtlasso(`*varlist*`,` *lasso_options*`)`	specify options for square-root lassos for variables in *varlist*; may be repeated
`reestimate`	refit the model after using `lassoselect` to select a different λ^*
`noheader`	do not display the header on the coefficient table
`coeflegend`	display legend instead of statistics

*`controls()` is required.

varsofinterest, *alwaysvars*, and *othervars* may contain factor variables. Base levels of factor variables cannot be set for *alwaysvars* and *othervars*. See [U] **11.4.3 Factor variables**.

`collect` is allowed; see [U] **11.1.10 Prefix commands**.

`reestimate`, `noheader`, and `coeflegend` do not appear in the dialog box.

See [U] **20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

`controls(`[`(`*alwaysvars*`)`] *othervars*`)` specifies the set of control variables, which control for omitted variables. Control variables are also known as confounding variables. `poregress` fits lassos for *depvar* and each of the *varsofinterest*. *alwaysvars* are variables that are always to be included in these lassos. *alwaysvars* are optional. *othervars* are variables that each lasso will choose to include or exclude. That is, each lasso will select a subset of *othervars*. The selected subset of *othervars* may differ across lassos. `controls()` is required.

`selection(plugin` | `cv` | `adaptive` | `bic)` specifies the selection method for choosing an optimal value of the lasso penalty parameter λ^* for each lasso or square-root lasso estimation. Separate lassos are estimated for *depvar* and each variable in *varsofinterest*. Specifying `selection()` changes the selection method for all of these lassos. You can specify different selection methods for different lassos using the option `lasso()` or `sqrtlasso()`. When `lasso()` or `sqrtlasso()` is used to specify a different selection method for the lassos of some variables, they override the global setting made using `selection()` for the specified variables.

`selection(plugin)` is the default. It selects λ^* based on a "plugin" iterative formula dependent on the data. See [LASSO] **lasso options**.

`selection(cv)` selects the λ^* that gives the minimum of the CV function. See [LASSO] **lasso options**.

`selection(adaptive)` selects λ^* using the adaptive lasso selection method. It cannot be specified when `sqrtlasso` is specified. See [LASSO] **lasso options**.

`selection(bic)` selects the λ^* that gives the minimum of the BIC function. See [LASSO] **lasso options**.

`sqrtlasso` specifies that square-root lassos be done rather than regular lassos. The option `lasso()` can be used with `sqrtlasso` to specify that regular lasso be done for some variables, overriding the global `sqrtlasso` setting for these variables. See [LASSO] **lasso options**.

`semi` specifies that the semipartialing-out lasso regression estimator be used instead of the fully partialing-out lasso estimator, which is the default. See *Methods and formulas* in [LASSO] **poregress**.

`missingok` specifies that, after fitting lassos, the estimation sample be redefined based on only the nonmissing observations of variables in the final model. In all cases, any observation with missing values for *depvar*, *varsofinterest*, *alwaysvars*, and *othervars* is omitted from the estimation sample for the lassos. By default, the same sample is used for calculation of the coefficients of the *varsofinterest* and their standard errors.

When `missingok` is specified, the initial estimation sample is the same as the default, but the sample used for the calculation of the coefficients of the *varsofinterest* can be larger. Now observations with missing values for any *othervars* not selected will be added to the estimation sample (provided there are no missing values for any of the variables in the final model).

missingok may produce more efficient estimates when data are missing completely at random. It does, however, have the consequence that estimation samples can change when selected variables differ in models fit using different selection methods. That is, when *othervars* contain missing values, the estimation sample for a model fit using the default selection(plugin) will likely differ from the estimation sample for a model fit using, for example, selection(cv).

SE/Robust

vce(*vcetype*) specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (robust) and that allow for intragroup correlation (cluster *clustvar*); see [R] ***vce_option***.

When vce(cluster *clustvar*) is specified, all lassos also account for clustering. For each lasso, this affects how the log-likelihood function is computed and how the sample is split in cross-validation; see *Methods and formulas* in [LASSO] **lasso**. Specifying vce(cluster *clustvar*) may lead to different selected controls and therefore to different point estimates for your variable of interest when compared to the estimation that ignores clustering.

Reporting

level(#); see [R] **Estimation options**.

display_options: noci, nopvalues, noomitted, vsquish, noemptycells, baselevels, allbaselevels, nofvlabel, fvwrap(#), fvwrapon(*style*), cformat(%*fmt*), pformat(%*fmt*), sformat(%*fmt*), and nolstretch; see [R] **Estimation options**.

Optimization

[no]log displays or suppresses a log showing the progress of the estimation. By default, one-line messages indicating when each lasso estimation begins are shown. Specify verbose to see a more detailed log.

verbose displays a verbose log showing the iterations of each lasso estimation. This option is useful when doing selection(cv) or selection(adaptive). It allows you to monitor the progress of the lasso estimations for these selection methods, which can be time consuming when there are many *othervars* specified in controls().

rseed(#) sets the random-number seed. This option can be used to reproduce results for selection(cv) and selection(adaptive). The default selection method selection(plugin) does not use random numbers. rseed(#) is equivalent to typing set seed # prior to running poregress. See [R] **set seed**.

Advanced

lasso(*varlist*, *lasso_options*) lets you set different options for different lassos, or advanced options for all lassos. You specify a *varlist* followed by the options you want to apply to the lassos for these variables. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. _all or * may be used to specify *depvar* and all *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are selection(...), grid(...), stop(#), tolerance(#), dtolerance(#), and cvtolerance(#). When lasso(*varlist*, selection(...)) is specified, it overrides any global selection() option for the variables in *varlist*. It also overrides the global sqrtlasso option for these variables. See [LASSO] **lasso options**.

sqrtlasso(*varlist*, *lasso_options*) works like the option lasso(), except square-root lassos for the variables in *varlist* are done rather than regular lassos. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are selection(...), grid(...), stop(#), tolerance(#), dtolerance(#), and cvtolerance(#). When sqrtlasso(*varlist*, selection(...)) is specified, it overrides any global selection() option for the variables in *varlist*. See [LASSO] **lasso options**.

The following options are available with poregress but are not shown in the dialog box:

reestimate is an advanced option that refits the poregress model based on changes made to the underlying lassos using lassoselect. After running poregress, you can select a different λ^* for one or more of the lassos estimated by poregress. After selecting λ^*, you type poregress, reestimate to refit the poregress model based on the newly selected λ's.

reestimate may be combined only with reporting options.

noheader prevents the coefficient table header from being displayed.

coeflegend; see [R] **Estimation options**.

Remarks and examples

poregress performs partialing-out lasso linear regression. This command estimates coefficients, standard errors, and confidence intervals and performs tests for variables of interest while using lassos to select from among potential control variables.

The linear regression model is

$$\mathbf{E}[y|\mathbf{d},\, \mathbf{x}] = \mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}'$$

where **d** are the variables for which we wish to make inferences and **x** are the potential control variables from which the lassos select. poregress reports estimated coefficients for $\boldsymbol{\alpha}$. However, partialing-out does not provide estimates of the coefficients on the control variables ($\boldsymbol{\beta}$) or their standard errors. No estimation results can be reported for $\boldsymbol{\beta}$.

For an introduction to the partialing-out lasso method for inference, as well as the double-selection and cross-fit partialing-out methods, see [LASSO] **Lasso inference intro**.

Examples that demonstrate how to use poregress and the other lasso inference commands are presented in [LASSO] **Inference examples**. In particular, we recommend reading *1 Overview* for an introduction to the examples and to the vl command, which provides tools for working with the large lists of variables that are often included when using lasso methods. See *2 Fitting and interpreting inferential models* for examples of fitting inferential lasso linear models and comparisons of the different methods available in Stata.

If you are interested in digging deeper into the lassos that are used to select controls, see *5 Exploring inferential model lassos* in [LASSO] **Inference examples**.

Stored results

poregress stores the following in e():

Scalars	
e(N)	number of observations
e(N_clust)	number of clusters
e(k_varsofinterest)	number of variables of interest
e(k_controls)	number of potential control variables
e(k_controls_sel)	number of selected control variables
e(df)	degrees of freedom for test of variables of interest
e(chi2)	χ^2
e(p)	*p*-value for test of variables of interest
e(rank)	rank of e(V)
Macros	
e(cmd)	poregress
e(cmdline)	command as typed
e(depvar)	name of dependent variable
e(lasso_depvars)	names of dependent variables for all lassos
e(varsofinterest)	variables of interest
e(controls)	potential control variables
e(controls_sel)	selected control variables
e(model)	linear
e(title)	title in estimation output
e(clustvar)	name of cluster variable
e(chi2type)	Wald; type of χ^2 test
e(vce)	*vcetype* specified in vce()
e(vcetype)	title used to label Std. err.
e(rngstate)	random-number state used
e(properties)	b V
e(predict)	program used to implement predict
e(select_cmd)	program used to implement lassoselect
e(marginsnotok)	predictions disallowed by margins
e(asbalanced)	factor variables fvset as asbalanced
e(asobserved)	factor variables fvset as asobserved
Matrices	
e(b)	coefficient vector
e(V)	variance–covariance matrix of the estimators
Functions	
e(sample)	marks estimation sample

In addition to the above, the following is stored in r():

Matrices	
r(table)	matrix containing the coefficients with their standard errors, test statistics, *p*-values, and confidence intervals

Note that results stored in r() are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

poregress implements two methods for the partialing-out lasso regression. We call the default method partialing-out lasso regression (POLR). We call the optional method, obtained by specifying option semi, a semipartialing-out lasso regression (SPOLR). We refer to these methods as versions of partialing-out regression because they reduce to the classic method of partialing-out regression in a special case discussed below.

The POLR was derived by Belloni et al. (2012) and Chernozhukov, Hansen, and Spindler (2015a, 2015b). The SPOLR was derived by Belloni et al. (2012), Belloni, Chernozhukov, and Hansen (2014), Belloni, Chernozhukov, and Kato (2015), and Belloni, Chernozhukov, and Wei (2016).

The authors referred to their methods as "instrumental-variable methods". We refer to these methods as partialing-out regression methods because the idea of partialing-out regression is more cross-disciplinary and because these methods do not need outside instrumental variables when the covariates are exogenous.

Mechanically, the POLR and the SPOLR methods are method of moments estimators in which the moment conditions are the score equations from an ordinary least-squares (OLS) estimator of a partial outcome on one or more partial covariates. The partial outcome is the residual from a regression of the outcome on the controls selected by a lasso. Each of the partial covariates is a residual from a regression of the covariate on the controls selected by a lasso.

The POLR method is limited to a linear model for the outcome. This method follows from Chernozhukov, Hansen, and Spindler (2015a; 2015b, sec. 5) and Chernozhukov et al. (2018, C18). The algorithm described in Chernozhukov, Hansen, and Spindler (2015a, 2015b) is for endogenous variables with many outside instruments and many controls. As they note, imposing an exogeneity assumption and assuming that there are no outside instruments reduces their algorithm to one for exogenous covariates with many controls.

The SPOLR method extends naturally to nonlinear models for the outcome and has two sources. It is implied by Belloni, Chernozhukov, and Kato (2015, algorithm 1), which is for a median regression of y on $\mathbf{x}$. Replacing median regression with mean regression in their step (i) and replacing the median moment condition with the mean moment condition in step (iii) produces the SPOLR algorithm detailed below. This algorithm is also implied by Belloni, Chernozhukov, and Wei (2016, table 1 and sec. 2.1) for a linear model.

The regression model is

$$\mathbf{E}[y|\mathbf{d},\, \mathbf{x}] = \mathbf{d}\boldsymbol{\alpha}' + \beta_0 + \mathbf{x}\boldsymbol{\beta}'$$

where $\mathbf{d}$ contains the J covariates of interest and $\mathbf{x}$ contains the p controls. The number of covariates in $\mathbf{d}$ must be small and fixed. The number of controls in $\mathbf{x}$ can be large and, in theory, can grow with the sample size; however, the number of nonzero elements in $\boldsymbol{\beta}$ must not be too large, which is to say that the model must be sparse.

POLR algorithm

1. For $j = 1, \ldots, J$, perform a linear lasso of d_j on $\mathbf{x}$, and denote the selected controls by $\tilde{\mathbf{x}}_j$.

   Each of these lassos can choose the lasso penalty parameter (λ_j^*) using the plugin estimator, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

2. For $j = 1, \ldots, J$, fit a linear regression of d_j on $\tilde{\mathbf{x}}_j$, denote the estimated coefficients by $\widehat{\gamma}_j$, and define the partial-covariate variable $z_j = d_j - \tilde{\mathbf{x}}_j\widehat{\gamma}_j$, with its ith observation denoted by $z_{j,i}$.

   Collect the J partial covariates for the ith observation into the vector $\mathbf{z}_i = (z_{1,i}, \ldots, z_{J,i})$.

3. Perform a linear lasso of y on $\mathbf{x}$ to select covariates $\tilde{\mathbf{x}}_y$.

   This lasso can choose the lasso penalty parameter (λ^*) using the plugin estimator, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

4. Fit a linear regression of y on $\tilde{\mathbf{x}}_y$, denote the estimated coefficients by $\widetilde{\boldsymbol{\beta}}_y$, and define the partial outcome for the ith observation by $\tilde{y}_i = y_i - \tilde{\mathbf{x}}_{y,i}\widetilde{\boldsymbol{\beta}}_y$.

5. Compute $\widehat{\boldsymbol{\alpha}}$ by solving the following J sample-moment equations.

$$\frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - \mathbf{z}_i\boldsymbol{\alpha}')\mathbf{z}_i' = \mathbf{0}$$

   The VCE is estimated by the robust estimator for method of moments.

SPOLR algorithm

1. For $j = 1, \ldots, J$, perform a linear lasso of d_j on $\mathbf{x}$, and denote the selected controls by $\tilde{\mathbf{x}}_j$.

   Each of these lassos can choose the lasso penalty parameter (λ_j^*) using the plugin estimator, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.
2. For $j = 1, \ldots, J$, fit a linear regression of d_j on $\tilde{\mathbf{x}}_j$, denote the estimated coefficients by $\widehat{\boldsymbol{\gamma}}_j$, and define the partial-covariate variable $z_j = d_j - \tilde{\mathbf{x}}_j\widehat{\boldsymbol{\gamma}}_j$, with its ith observation denoted by $z_{j,i}$.

   Collect the J partial covariates for the ith observation into the vector $\mathbf{z}_i = (z_{1,i}, \ldots, z_{J,i})$.
3. Perform a linear lasso of y on $\mathbf{d}$ and $\mathbf{x}$ to select covariates $\check{\mathbf{x}}_y$.

   This lasso can choose the lasso penalty parameter (λ^*) using the plugin estimator, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.
4. Fit a linear regression of y on $\mathbf{d}$ and $\check{\mathbf{x}}_y$, let $\check{\boldsymbol{\beta}}$ be the estimated coefficients on $\check{\mathbf{x}}_y$, and define the partial outcome for the ith observation by $\check{y}_i = y_i - \check{\mathbf{x}}_{y,i}\check{\boldsymbol{\beta}}'$.
5. Compute $\widehat{\boldsymbol{\alpha}}$ by solving the following J sample-moment equations.

$$\frac{1}{n}\sum_{i=1}^{n}(\check{y}_i - \mathbf{d}_i\boldsymbol{\alpha}')\mathbf{z}_i' = \mathbf{0}$$

   The VCE is estimated by the robust estimator for method of moments.

See *Methods and formulas* in [LASSO] **lasso** for details on how the lassos in steps 1 and 3 of both algorithms choose their penalty parameter (λ^*).

References

Belloni, A., D. Chen, V. Chernozhukov, and C. B. Hansen. 2012. Sparse models and methods for optimal instruments with an application to eminent domain. *Econometrica* 80: 2369–2429. https://doi.org/10.3982/ECTA9626.

Belloni, A., V. Chernozhukov, and C. B. Hansen. 2014. Inference on treatment effects after selection among high-dimensional controls. *Review of Economic Studies* 81: 608–650. https://doi.org/10.1093/restud/rdt044.

Belloni, A., V. Chernozhukov, and K. Kato. 2015. Uniform post-selection inference for least absolute deviation regression and other Z-estimation problems. *Biometrika* 102: 77–94. https://doi.org/10.1093/biomet/asu056.

Belloni, A., V. Chernozhukov, and Y. Wei. 2016. Post-selection inference for generalized linear models with many controls. *Journal of Business and Economic Statistics* 34: 606–619. https://doi.org/10.1080/07350015.2016.1166116.

Chernozhukov, V., D. Chetverikov, M. Demirer, E. Duflo, C. B. Hansen, W. K. Newey, and J. M. Robins. 2018. Double/debiased machine learning for treatment and structural parameters. *Econometrics Journal* 21: C1–C68. https://doi.org/10.1111/ectj.12097.

Chernozhukov, V., C. B. Hansen, and M. Spindler. 2015a. Post-selection and post-regularization inference in linear models with many controls and instruments. *American Economic Review* 105: 486–490. https://doi.org/10.1257/aer.p20151022.

———. 2015b. Valid post-selection and post-regularization inference: An elementary, general approach. *Annual Review of Economics* 7: 649–688. https://doi.org/10.1146/annurev-economics-012315-015826.

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **dsregress** — Double-selection lasso linear regression

[LASSO] **xporegress** — Cross-fit partialing-out lasso linear regression

[R] **regress** — Linear regression

[U] 20 Estimation and postestimation commands

sqrtlasso — Square-root lasso for prediction and model selection

Description

sqrtlasso selects covariates and fits linear models using square-root lasso. Results from sqrtlasso can be used for prediction and model selection. Results from sqrtlasso are typically similar to results from lasso.

sqrtlasso saves but does not display estimated coefficients. The [LASSO] **lasso postestimation** commands can be used to generate predictions, report coefficients, and display measures of fit.

For an introduction to lasso, see [LASSO] **Lasso intro**.

Quick start

Fit a linear model for y1, and select covariates from x1 to x100 using cross-validation (CV)

```
sqrtlasso y x1-x100
```

Same as above, but force x1 and x2 to be in the model while square-root lasso selects from x3 to x100

```
sqrtlasso y (x1 x2) x3-x100
```

Set a random-number seed for reproducibility

```
sqrtlasso y x1-x100, rseed(1234)
```

Calculate the CV function beyond the CV minimum to get the full coefficient paths, knots, etc.

```
sqrtlasso y x1-x100, selection(cv, alllambdas)
```

Menu

Statistics > Lasso > Square-root lasso

Syntax

sqrtlasso *depvar* [(*alwaysvars*)] *othervars* [*if*] [*in*] [*weight*] [, *options*]

alwaysvars are variables that are always included in the model.

othervars are variables that sqrtlasso will choose to include in or exclude from the model.

options	Description
Model	
noconstant	suppress constant term
selection(*sel_method*)	selection method to select a value of the square-root lasso penalty parameter λ^* from the set of possible λ's
offset(*varname*$_o$)	include *varname*$_o$ in model with coefficient constrained to 1
cluster(*clustvar*)	specify cluster variable *clustvar*
Optimization	
[no]log	display or suppress an iteration log
rseed(#)	set random-number seed
grid(#$_g$ [, ratio(#) min(#)])	specify the set of possible λ's using a logarithmic grid with #$_g$ grid points
stop(#)	tolerance for stopping the iteration over the λ grid early
cvtolerance(#)	tolerance for identification of the CV function minimum
bictolerance(#)	tolerance for identification of the BIC function minimum
tolerance(#)	convergence tolerance for coefficients based on their values
dtolerance(#)	convergence tolerance for coefficients based on deviance
penaltywt(*matname*)	programmer's option for specifying a vector of weights for the coefficients in the penalty term

sel_method	Description
cv [, *cv_opts*]	select λ^* using CV; the default
plugin [, *plugin_opts*]	select λ^* using a plugin iterative formula
bic [, *bic_opts*]	select λ^* using BIC function
none	do not select λ^*

cv_opts	Description
folds(#)	use # folds for CV
alllambdas	fit models for all λ's in the grid or until the stop(#) tolerance is reached; by default, the CV function is calculated sequentially by λ, and estimation stops when a minimum is identified
serule	use the one-standard-error rule to select λ^*
stopok	when the CV function does not have an identified minimum and the stop(#) stopping criterion for λ was reached at λ_{stop}, set the selected λ^* to be λ_{stop}; the default
strict	do not select λ^* when the CV function does not have an identified minimum; this is a stricter alternative to the default stopok
gridminok	when the CV function does not have an identified minimum and the stop(#) stopping criterion for λ was not reached, set the selected λ^* to be the minimum of the λ grid, λ_{gmin}; this is a looser alternative to the default stopok and is rarely used

plugin_opts	Description
heteroskedastic	assume model errors are heteroskedastic; the default
homoskedastic	assume model errors are homoskedastic

bic_opts	Description
alllambdas	fit models for all λ's in the grid or until the stop(#) tolerance is reached; by default, the BIC function is calculated sequentially by λ, and estimation stops when a minimum is identified
stopok	when the BIC function does not have an identified minimum and the stop(#) stopping criterion for λ was reached at λ_{stop}, set the selected λ^* to be λ_{stop}; the default
strict	do not select λ^* when the BIC function does not have an identified minimum; this is a stricter alternative to the default stopok
gridminok	when the BIC function does not have an identified minimum and the stop(#) stopping criterion for λ was not reached, set the selected λ^* to be the minimum of the λ grid, λ_{gmin}; this is a looser alternative to the default stopok and is rarely used
postselection	use postselection coefficients to compute BIC

alwaysvars and *othervars* may contain factor variables; see **[U] 11.4.3 Factor variables**.

collect is allowed; see **[U] 11.1.10 Prefix commands**.

Default weights are not allowed. iweights are allowed with all *sel_method* options. See **[U] 11.1.6 weight**.

penaltywt(*matname*) does not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

See [LASSO] **lasso fitting** for an overview of the lasso estimation procedure and a detailed description of how to set options to control it.

Model

noconstant omits the constant term. Note, however, when there are factor variables among the *othervars*, sqrtlasso can potentially create the equivalent of the constant term by including all levels of a factor variable. This option is likely best used only when all the *othervars* are continuous variables and there is a conceptual reason why there should be no constant term.

selection(cv), selection(plugin), selection(bic), and selection(none) specify the selection method used to select λ^*. These options also allow suboptions for controlling the specified selection method.

selection(cv [, *cv_opts*]) is the default. It selects λ^* to be the λ that gives the minimum of the CV function. lasso postestimation commands can be used after selection(cv) to assess alternative λ^* values.

cv_opts are folds(#), alllambdas, serule, stopok, strict, and gridminok.

folds(#) specifies that CV with # folds be done. The default is folds(10).

alllambdas specifies that models be fit for all λ's in the grid or until the stop(#) tolerance is reached. By default, models are calculated sequentially from largest to smallest λ, and the CV function is calculated after each model is fit. If a minimum of the CV function is found, the computation ends at that point without evaluating additional smaller λ's.

alllambdas computes models for these additional smaller λ's. Because computation time is greater for smaller λ, specifying alllambdas may increase computation time manyfold. Specifying alllambdas is typically done only when a full plot of the CV function is wanted for assurance that a true minimum has been found. Regardless of whether alllambdas is specified, the selected λ^* will be the same.

serule selects λ^* based on the "one-standard-error rule" recommended by Hastie, Tibshirani, and Wainwright (2015, 13–14) instead of the λ that minimizes the CV function. The one-standard-error rule selects the largest λ for which the CV function is within a standard error of the minimum of the CV function.

stopok, strict, and gridminok specify what to do when the CV function does not have an identified minimum. A minimum is identified at λ^* when the CV function at both larger and smaller adjacent λ's is greater than it is at λ^*. When the CV function has an identified minimum, these options all do the same thing: the selected λ^* is the λ that gives the minimum. In some cases, however, the CV function declines monotonically as λ gets smaller and never rises to identify a minimum. When the CV function does not have an identified minimum, stopok and gridminok make alternative selections for λ^*, and strict makes no selection. You may specify only one of stopok, strict, or gridminok; stopok is the default if you do not specify one. With each of these options, estimation results are always left in place, and alternative λ^* can be selected and evaluated.

stopok specifies that when the CV function does not have an identified minimum and the stop(#) stopping tolerance for λ was reached, the selected λ^* is λ_{stop}, the λ that met the stopping criterion. λ_{stop} is the smallest λ for which coefficients are estimated, and it is assumed that λ_{stop} has a CV function value close to the true minimum. When no minimum is identified and the stop(#) criterion is not met, an error is issued.

strict requires the CV function to have an identified minimum, and if not, an error is issued.

gridminok is a rarely used option that specifies that when the CV function has no identified minimum and the stop(#) stopping criterion was not met, λ_{gmin}, the minimum of the λ grid, is the selected λ^*.

The gridminok selection criterion is looser than the default stopok, which is looser than strict. With strict, only an identified minimum is selected. With stopok, either the identified minimum or λ_{stop} is selected. With gridminok, either the identified minimum or λ_{stop} or λ_{gmin} is selected, in this order.

selection(plugin [, *plugin_opts*]) selects λ^* based on a "plugin" iterative formula dependent on the data. The plugin method was designed for lasso inference methods and is useful when using sqrtlasso to manually implement inference methods, such as double-selection lasso. The plugin estimator calculates a value for λ^* that dominates the noise in the estimating equations, which makes it less likely to include variables that are not in the true model. See *Methods and formulas*.

selection(plugin) does not estimate coefficients for any other values of λ, so it does not require a λ grid, and none of the grid options apply. It is much faster than selection(cv) because estimation is done only for a single value of λ. It is an iterative procedure, however, and if the plugin is computing estimates for a small λ (which means many nonzero coefficients), the estimation can still be time consuming. Because estimation is done only for one λ, you cannot assess alternative λ^* as the other selection methods allow.

plugin_opts are heteroskedastic and homoskedastic.

`heteroskedastic` assumes model errors are heteroskedastic. It is the default. Specifying `selection(plugin)` is equivalent to specifying `selection(plugin, heteroskedastic)`.

`homoskedastic` assumes model errors are homoskedastic. See *Methods and formulas*.

`selection(bic` [, *bic_opts*]`)` selects λ^* by using the Bayesian information criterion function. It selects the λ^* with the minimum BIC function value.

bic_opts are `alllambdas`, `stopok`, `strict`, `gridminok`, and `postselection`.

`alllambdas` specifies that models be fit for all λ's in the grid or until the `stop(#)` tolerance is reached. By default, models are calculated sequentially from largest to smallest λ, and the BIC function is calculated after each model is fit. If a minimum of the BIC function is found, the computation ends at that point without evaluating additional smaller λ's.

`alllambdas` computes models for these additional smaller λ's. Because computation time is greater for smaller λ, specifying `alllambdas` may increase computation time manyfold. Specifying `alllambdas` is typically done only when a full plot of the BIC function is wanted for assurance that a true minimum has been found. Regardless of whether `alllambdas` is specified, the selected λ^* will be the same.

`stopok`, `strict`, and `gridminok` specify what to do when the BIC function does not have an identified minimum. A minimum is identified at λ^* when the BIC function at both larger and smaller adjacent λ's is greater than it is at λ^*. When the BIC function has an identified minimum, these options all do the same thing: the selected λ^* is the λ that gives the minimum. In some cases, however, the BIC function declines monotonically as λ gets smaller and never rises to identify a minimum. When the BIC function does not have an identified minimum, `stopok` and `gridminok` make alternative selections for λ^*, and `strict` makes no selection. You may specify only one of `stopok`, `strict`, or `gridminok`; `stopok` is the default if you do not specify one. With each of these options, estimation results are always left in place, and alternative λ^* can be selected and evaluated.

`stopok` specifies that when the BIC function does not have an identified minimum and the `stop(#)` stopping tolerance for λ was reached, the selected λ^* is λ_{stop}, the λ that met the stopping criterion. λ_{stop} is the smallest λ for which coefficients are estimated, and it is assumed that λ_{stop} has a BIC function value close to the true minimum. When no minimum is identified and the `stop(#)` criterion is not met, an error is issued.

`strict` requires the BIC function to have an identified minimum, and if not, an error is issued.

`gridminok` is a rarely used option that specifies that when the BIC function has no identified minimum and the `stop(#)` stopping criterion was not met, then λ_{gmin}, the minimum of the λ grid, is the selected λ^*.

The `gridminok` selection criterion is looser than the default `stopok`, which is looser than `strict`. With `strict`, only an identified minimum is selected. With `stopok`, either the identified minimum or λ_{stop} is selected. With `gridminok`, either the identified minimum or λ_{stop} or λ_{gmin} is selected, in this order.

`postselection` specifies to use the postselection coefficients to compute the BIC function. By default, the penalized coefficients are used.

selection(none) does not select a λ^*. Square-root lasso is estimated for the grid of values for λ, but no attempt is made to determine which λ should be selected. The postestimation command lassoknots can be run to view a table of λ's that define the knots (the sets of nonzero coefficients) for the estimation. The lassoselect command can be used to select a value for λ^*, and lassogof can be run to evaluate the prediction performance of λ^*.

When selection(none) is specified, the CV function is not computed. If you want to view the knot table with values of the CV function shown and then select λ^*, you must specify selection(cv). There are no suboptions for selection(none).

offset(*varname*$_o$) specifies that *varname*$_o$ be included in the model with its coefficient constrained to be 1.

cluster(*clustvar*) specifies the cluster variable *clustvar*. Specifying a cluster variable will affect how the log-likelihood function is computed and the sample split in cross-validation. The log-likelihood function is computed as the sum of the log likelihood at the cluster levels. If option selection(cv) is specified, the cross-validation sample is split by the clusters defined by *clustvar*. That is, the subsample in each fold is drawn on the cluster level. Therefore, all observations in a cluster are kept together in the same subsample.

Optimization

[no]log displays or suppresses a log showing the progress of the estimation.

rseed(#) sets the random-number seed. This option can be used to reproduce results for selection(cv). The other selection methods, selection(plugin) and selection(none), do not use random numbers. rseed(#) is equivalent to typing set seed # prior to running sqrtlasso. See [R] **set seed**.

grid(#$_g$ [, ratio(#) min(#)]) specifies the set of possible λ's using a logarithmic grid with #$_g$ grid points.

#$_g$ is the number of grid points for λ. The default is #$_g$ = 100. The grid is logarithmic with the ith grid point ($i = 1, \dots, n = \#_g$) given by $\ln \lambda_i = [(i-1)/(n-1)] \ln r + \ln \lambda_{\text{gmax}}$, where $\lambda_{\text{gmax}} = \lambda_1$ is the maximum, $\lambda_{\text{gmin}} = \lambda_n =$ min(#) is the minimum, and $r = \lambda_{\text{gmin}}/\lambda_{\text{gmax}} =$ ratio(#) is the ratio of the minimum to the maximum.

ratio(#) specifies $\lambda_{\text{gmin}}/\lambda_{\text{gmax}}$. The maximum of the grid, λ_{gmax}, is set to the smallest λ for which all the coefficients in the lasso are estimated to be zero (except the coefficients of the *alwaysvars*). λ_{gmin} is then set based on ratio(#). When $p < N$, where p is the total number of *othervars* and *alwaysvars* (not including the constant term) and N is the number of observations, the default value of ratio(#) is 1e−4. When $p \geq N$, the default is 1e−2.

min(#) sets λ_{gmin}. By default, λ_{gmin} is based on ratio(#) and λ_{gmax}, which is computed from the data.

stop(#) specifies a tolerance that is the stopping criterion for the λ iterations. The default is 1e−5. This option does not apply when the selection method is selection(plugin). Estimation starts with the maximum grid value, λ_{gmax}, and iterates toward the minimum grid value, λ_{gmin}. When the relative difference in the deviance produced by two adjacent λ grid values is less than stop(#), the iteration stops and no smaller λ's are evaluated. The value of λ that meets this tolerance is denoted by λ_{stop}. Typically, this stopping criterion is met before the iteration reaches λ_{gmin}.

Setting stop(#) to a larger value means that iterations are stopped earlier at a larger λ_{stop}. To produce coefficient estimates for all values of the λ grid, you can specify stop(0). Note, however, that computations for small λ's can be extremely time consuming. In terms of time, when you use selection(cv), the optimal value of stop(#) is the largest value that allows estimates for just enough λ's to be computed to identify the minimum of the CV function. When setting stop(#) to larger values, be aware of the consequences of the default λ^* selection procedure given by the default stopok. You may want to override the stopok behavior by using strict.

cvtolerance(#) is a rarely used option that changes the tolerance for identifying the minimum CV function. For linear models, a minimum is identified when the CV function rises above a nominal minimum for at least three smaller λ's with a relative difference in the CV function greater than #. For nonlinear models, at least five smaller λ's are required. The default is 1e−3. Setting # to a bigger value makes a stricter criterion for identifying a minimum and brings more assurance that a declared minimum is a true minimum, but it also means that models may need to be fit for additional smaller λ, which can be time consuming. See *Methods and formulas* for [LASSO] **lasso** for more information about this tolerance and the other tolerances.

bictolerance(#) is a rarely used option that changes the tolerance for identifying the minimum BIC function. A minimum is identified when the BIC function rises above a nominal minimum for at least two smaller λ's with a relative difference in the BIC function greater than #. The default is 1e−2. Setting # to a bigger value makes a stricter criterion for identifying a minimum and brings more assurance that a declared minimum is a true minimum, but it also means that models may need to be fit for additional smaller λ, which can be time consuming. See *Methods and formulas* in [LASSO] **lasso** for more information about this tolerance and the other tolerances.

tolerance(#) is a rarely used option that specifies the convergence tolerance for the coefficients. Convergence is achieved when the relative change in each coefficient is less than this tolerance. The default is tolerance(1e-7).

dtolerance(#) is a rarely used option that changes the convergence criterion for the coefficients. When dtolerance(#) is specified, the convergence criterion is based on the change in deviance instead of the change in the values of coefficient estimates. Convergence is declared when the relative change in the deviance is less than #. More-accurate coefficient estimates are typically achieved by not specifying this option and instead using the default tolerance(1e-7) criterion or specifying a smaller value for tolerance(#).

The following option is available with sqrtlasso but is not shown in the dialog box:

penaltywt(*matname*) is a programmer's option for specifying a vector of weights for the coefficients in the penalty term. The contribution of each coefficient to the square-root lasso penalty term is multiplied by its corresponding weight. Weights must be nonnegative. By default, each coefficient's penalty weight is 1.

Remarks and examples

We assume you have read the lasso introduction [LASSO] **Lasso intro**.

The square-root lasso is an alternative version of lasso. Lasso minimizes

$$\frac{1}{2N}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}')'(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}') + \lambda\sum_{j=1}^{p}|\beta_j|$$

whereas square-root lasso minimizes

$$\sqrt{\frac{1}{N}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}')'(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}')} + \frac{\lambda}{N}\sum_{j=1}^{p}|\beta_j|$$

In the square-root formulation, the standard deviation of the error term becomes a multiplicative constant that drops out of the minimization. This lack of dependence facilitates the derivation of plugin estimators for the lasso penalty parameter λ^* because there is no need to estimate the standard deviation of the error term as part of the plugin formula.

Square-root lasso is primarily used in combination with a plugin estimator for λ^*. The resulting square-root lasso estimation can be used with the double-selection or partialing-out methods described in [LASSO] **Lasso inference intro**.

Square-root lasso can also be used on its own for prediction or model selection. To be consistent with `lasso`, the default selection method for λ^* is CV. To use the plugin estimator, specify the option `selection(plugin)`.

Square-root lasso was formulated by Belloni, Chernozhukov, and Wang (2011), who also derived the square-root lasso plugin estimator for λ, which is implemented here.

▷ Example 1: Square-root lasso and lasso

Let's compare square-root lasso with an ordinary lasso to illustrate that their results are numerically similar when used with CV.

We load the example dataset we used in [LASSO] **lasso examples**. It has stored variable lists created by `vl`. See [D] **vl** for a complete description of the `vl` system and how to use it to manage large variable lists.

After we load the dataset, we type `vl rebuild` to make the saved variable lists active again.

```
. use https://www.stata-press.com/data/r19/fakesurvey_vl
(Fictitious survey data with vl)
. vl rebuild
Rebuilding vl macros ...
─────────────────────────────────────────────────────────────────────────────────
                  │                    Macro's contents
                  │──────────────────────────────────────────────────────────────
Macro             │  # Vars   Description
──────────────────┼──────────────────────────────────────────────────────────────
System            │
  $vldummy        │      98   0/1 variables
  $vlcategorical  │      16   categorical variables
  $vlcontinuous   │      29   continuous variables
  $vluncertain    │      16   perhaps continuous, perhaps categorical variables
  $vlother        │      12   all missing or constant variables
User              │
  $demographics   │       4   variables
  $factors        │     110   variables
  $idemographics  │           factor-variable list
  $ifactors       │           factor-variable list
─────────────────────────────────────────────────────────────────────────────────
```

We randomly split our data into two samples of equal sizes. One we will fit lassos on, and the other we will use to test their predictions. We use `splitsample` to generate a variable indicating the samples.

```
. set seed 1234
. splitsample, generate(sample) nsplit(2)
. label define svalues 1 "Training" 2 "Testing"
. label values sample svalues
```

We have four user-defined variable lists, `demographics`, `factors`, `idemographics`, and `ifactors`. The variable lists `idemographics` and `ifactors` contain factor-variable versions of the categorical variables in `demographics` and `factors`. That is, a variable `q3` in `demographics` is `i.q3` in `idemographics`. See the examples in [LASSO] **lasso examples** to see how we created these variable lists.

We are going to use `idemographics` and `ifactors` along with the system-defined variable list `vlcontinuous` as arguments to `sqrtlasso`. Together they contain the potential variables we want to specify. Variable lists are actually global macros, and when we use them as arguments in commands, we put a $ in front of them.

We also set the random-number seed using the `rseed()` option so we can reproduce our results.

```
. sqrtlasso q104 $idemographics $ifactors $vlcontinuous if sample == 1,
> rseed(1234)
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = 104.6235   no. of nonzero coef. =   0
Folds: 1...5....10   CVF =  17.2848
  (output omitted)
Grid value 23:    lambda = 13.51264   no. of nonzero coef. =  87
Folds: 1...5....10   CVF = 12.35321
... cross-validation complete ... minimum found
Square-root lasso linear model              No. of obs        =        458
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds   =         10

─────────────────────────────────────────────────────────────────────────
        │                                No. of      Out-of-      CV mean
        │                               nonzero       sample   prediction
     ID │     Description      lambda     coef.    R-squared        error
────────┼────────────────────────────────────────────────────────────────
      1 │    first lambda    104.6235         0      -0.0058      17.2848
     17 │   lambda before    23.61373        53       0.2890     12.21892
   * 18 │ selected lambda    21.51595        61       0.2901     12.19933
     19 │    lambda after    19.60453        67       0.2899     12.20295
     23 │     last lambda    13.51264        87       0.2812     12.35321
─────────────────────────────────────────────────────────────────────────
* lambda selected by cross-validation.
. estimates store sqrtcv
```

The square-root lasso with the default CV selection method selected a model with 61 variables in it.

Let's run lasso with the same potential variables.

```
. lasso linear q104 $idemographics $ifactors $vlcontinuous if sample == 1,
> rseed(1234)
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .9469819   no. of nonzero coef. =   0
  (output omitted)
Grid value 25:    lambda = .1015418   no. of nonzero coef. =  78
Folds: 1...5....10   CVF = 12.26768
... cross-validation complete ... minimum found
Lasso linear model                          No. of obs        =        458
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds   =         10

                                              No. of      Out-of-      CV mean
                                             nonzero       sample   prediction
       ID     Description         lambda       coef.    R-squared        error

        1     first lambda      .9469819           0      -0.0046     17.26383
       19     lambda before     .1774471          47       0.2899     12.20399
     * 20   selected lambda     .1616832          51       0.2912     12.18122
       21      lambda after     .1473197          60       0.2908     12.18739
       25       last lambda     .1015418          78       0.2862     12.26768

* lambda selected by cross-validation.
. estimates store lassocv
```

Lasso selected a model with 51 variables in it.

After we ran sqrtlasso and lasso, we used estimates store to keep the results in memory. This lets us compare the models. We can use lassocoef to view the coefficient estimates. We display the standardized coefficients and sort them so that the biggest in absolute values are shown first.

```
. lassocoef sqrtcv lassocv, display(coef, standardized) sort(coef, standardized)

                   sqrtcv     lassocv

         q19
          No    -.8446332   -.8119414

         q85
          No    -.7089993   -.6940387
      3.q156    -.6843823   -.6727969

        q101
          No     .5981556    .5785246

         q48
          No    -.5867942   -.5502145

         q88
          No     .5793049     .553872

         q38
           4    -.5275709   -.5089004

          q5
          No    -.4795077    -.467305
         q22    -.4610605   -.4410858
         q31     .4556527    .4047143
```

```
             |
         q56 |
          No |  -.4482692  -.4026312
        q139 |  -.4189969  -.4118033
             |
         q73 |
          No |  -.3565698  -.3368294
             |
         q96 |
          No |  -.3149921  -.2950566
       3.q16 |   -.263147  -.2278278
             |
         q43 |
          No |  -.2605833  -.2355772
             |
         q50 |
          No |   .2455526   .2307073
             |
        q149 |
          No |  -.2407299  -.2070948
       2.q84 |  -.2321074  -.2150944
             |
        q109 |
          No |   .1965246   .1530308
             |
         q49 |
          No |   .1937052   .1626059
             |
        q159 |
          No |   .1870743   .1771646
             |
        q115 |
          No |    .153256   .1272736
      3.q134 |   .1525998   .1418469
             |
        q108 |
          No |  -.1491124  -.1469051
             |
         q91 |
          No |  -.1475877  -.1252736
             |
        q140 |
          No |   -.142592  -.1192079
       2.q34 |   .1397604   .1155922
         q93 |  -.1379424  -.0964044
             |
         q14 |
          No |  -.1377481  -.0964684
             |
      gender |
      Female |  -.1296337  -.1047897
             |
        q153 |
          No |   .1238655   .0835772
         q53 |   .1123144   .0813566
             |
         q65 |
           3 |   .1035524    .084643
             |
         q38 |
           3 |   .0922535    .086774
             |
        q160 |
          No  |  -.0901901  -.0763008
              |
           q3 |
          No  |   -.082771  -.0574645
          age |  -.0707354  -.0590426
              |
        q102 |
          No  |  -.0578734  -.0427812
              |
          q44 |
          No  |   .0561402   .0301015
      1.q110 |  -.0556488  -.0268615
              |
        q154 |
          No  |   .0492342   .0188979
              |
        q130 |
          No  |  -.0453674  -.0288351
          q18 |  -.0428028   -.018666
              |
          q97 |
          No  |   .0427896    .021222
              |
        q142 |
          No  |  -.0427358  -.0188524
              |
          q75 |
          No  |  -.0341663  -.0011199
        q111 |  -.0333302  -.0294021
       3.q95 |  -.0214817
              |
          q65 |
           4  |  -.0213682
              |
          q38 |
           2  |   .0197855
              |
       0.q74 |   .0165583
       0.q33 |   -.016441
          q20 |   .0147089
              |
          q94 |
          No  |   .0136563    .013323
          q52 |   .0132519
      0.q138 |  -.0125278
       0.q71 |    .012269
              |
          q13 |
          No  |   .0094304   .0027091
              |
        q105 |
       Fair  |   .0052163     .00026
       0.q59 |   .0036381
       _cons |          0          0
-----------------------------------

Legend:
  b - base level
  e - empty cell
  o - omitted
```

Numerically, the coefficients are similar. The six variables that square-root lasso selected—but lasso did not—are among the variables with the smallest coefficients.

We split the sample in half so we could look at the out-of-sample prediction. We use lassogof to do this using postselection coefficients.

```
. lassogof sqrtcv lassocv, over(sample) postselection
Postselection coefficients

-------------------------------------------------------------
Name              sample |         MSE    R-squared      Obs
-------------------------+-----------------------------------
sqrtcv                   |
                Training |    8.419174       0.5184      503
                 Testing |    15.09863       0.2402      487
-------------------------+-----------------------------------
lassocv                  |
                Training |    8.595046       0.5083      503
                 Testing |    14.66581       0.2600      491
-------------------------------------------------------------
```

Both square-root lasso and lasso did significantly worse predicting out of sample than they did in sample. This is typical in many cases when there are many variables with small coefficients in the models.

Let's compare the plugin estimators for both square-root lasso and lasso.

```
. sqrtlasso q104 $idemographics $ifactors $vlcontinuous, selection(plugin)
Computing plugin lambda ...
Iteration 1:     lambda = 134.4262   no. of nonzero coef. =   5
Iteration 2:     lambda = 134.4262   no. of nonzero coef. =   8
Iteration 3:     lambda = 134.4262   no. of nonzero coef. =   8
Square-root lasso linear model                 No. of obs        =        914
                                               No. of covariates =        277
Selection: Plugin heteroskedastic

--------------------------------------------------------------------------
        |                                No. of
        |                               nonzero    In-sample
     ID |     Description      lambda     coef.    R-squared          BIC
--------+-----------------------------------------------------------------
    * 1 |  selected lambda   134.4262         8       0.0835     5233.117
--------------------------------------------------------------------------
* lambda selected by plugin formula assuming heteroskedastic errors.
```

Square-root lasso with plugin selected only 8 variables. Let's see what lasso does.

```
. lasso linear q104 $idemographics $ifactors $vlcontinuous,
> selection(plugin) rseed(1234)
Computing plugin lambda ...
Iteration 1:     lambda = .1470747   no. of nonzero coef. =   8
Iteration 2:     lambda = .1470747   no. of nonzero coef. =  11
Iteration 3:     lambda = .1470747   no. of nonzero coef. =  13
Iteration 4:     lambda = .1470747   no. of nonzero coef. =  15
Iteration 5:     lambda = .1470747   no. of nonzero coef. =  15
Lasso linear model                          No. of obs        =        914
                                            No. of covariates =        277
Selection: Plugin heteroskedastic

--------------------------------------------------------------------------
         |                                No. of
         |                               nonzero    In-sample
      ID |     Description      lambda     coef.    R-squared        BIC
---------+----------------------------------------------------------------
     * 1 |  selected lambda   .1470747        15       0.1549   5206.721
--------------------------------------------------------------------------
* lambda selected by plugin formula assuming heteroskedastic errors.
```

Lasso with plugin selected a few more—15 variables in total. We can see from the in-sample R^2 that the predictive capabilities of models using plugin are much lower than those using CV. We expect this because plugin estimators were designed as a tool for inferential models, not for prediction.

◁

Stored results

sqrtlasso stores the following in e():

Scalars

e(N)	number of observations
e(N_clust)	number of clusters
e(k_allvars)	number of potential variables
e(k_nonzero_sel)	number of nonzero coefficients for selected model
e(k_nonzero_cv)	number of nonzero coefficients at CV mean function minimum
e(k_nonzero_serule)	number of nonzero coefficients for one-standard-error rule
e(k_nonzero_min)	minimum number of nonzero coefficients among estimated λ's
e(k_nonzero_max)	maximum number of nonzero coefficients among estimated λ's
e(k_nonzero_bic)	number of nonzero coefficients at BIC function minimum
e(lambda_sel)	value of selected λ^*
e(lambda_gmin)	value of λ at grid minimum
e(lambda_gmax)	value of λ at grid maximum
e(lambda_last)	value of last λ computed
e(lambda_cv)	value of λ at CV mean function minimum
e(lambda_serule)	value of λ for one-standard-error rule
e(lambda_bic)	value of λ at BIC function minimum
e(ID_sel)	ID of selected λ^*
e(ID_cv)	ID of λ at CV mean function minimum
e(ID_serule)	ID of λ for one-standard-error rule
e(ID_bic)	ID of λ at BIC function minimum
e(cvm_min)	minimum CV mean function value
e(cvm_serule)	CV mean function value at one-standard-error rule
e(devratio_min)	minimum deviance ratio
e(devratio_max)	maximum deviance ratio
e(L1_min)	minimum value of ℓ_1-norm of penalized unstandardized coefficients
e(L1_max)	maximum value of ℓ_1-norm of penalized unstandardized coefficients

`e(L2_min)`	minimum value of ℓ_2-norm of penalized unstandardized coefficients
`e(L2_max)`	maximum value of ℓ_2-norm of penalized unstandardized coefficients
`e(ll_sel)`	log-likelihood value of selected model
`e(n_lambda)`	number of λ's
`e(n_fold)`	number of CV folds
`e(stop)`	stopping rule tolerance
Macros	
`e(cmd)`	`sqrtlasso`
`e(cmdline)`	command as typed
`e(depvar)`	name of dependent variable
`e(allvars)`	names of all potential variables
`e(allvars_sel)`	names of all selected variables
`e(alwaysvars)`	names of always-included variables
`e(othervars_sel)`	names of other selected variables
`e(post_sel_vars)`	all variables needed for post-square-root lasso
`e(clustvar)`	name of cluster variable
`e(lasso_selection)`	selection method
`e(sel_criterion)`	criterion used to select λ^*
`e(plugin_type)`	type of plugin λ
`e(model)`	`linear`, `logit`, `poisson`, or `probit`
`e(title)`	title in estimation output
`e(rngstate)`	random-number state used
`e(properties)`	`b`
`e(predict)`	program used to implement `predict`
`e(marginsnotok)`	predictions disallowed by `margins`
Matrices	
`e(b)`	penalized unstandardized coefficient vector
`e(b_standardized)`	penalized standardized coefficient vector
`e(b_postselection)`	postselection coefficient vector
Functions	
`e(sample)`	marks estimation sample

In addition to the above, the following is stored in `r()`:

Matrices	
`r(table)`	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

This section provides the methods and formulas for the methods implemented in `sqrtlasso`. The square-root lasso was derived by Belloni and Chernozhukov (2011).

Methods and formulas are presented under the following headings:

Notation

`sqrtlasso` estimates the parameters by finding the minimum of a penalized objective function. The penalized objective function is

$$Q = \sqrt{\frac{1}{N}\sum_{i=1}^{N} w_i(y_i - \beta_0 - \mathbf{x}_i\boldsymbol{\beta}')^2} + \frac{\lambda}{N}\sum_{j=1}^{p}\kappa_j|\beta_j| \tag{1}$$

where N is the number of observations, w_i are observation-level weights, β_0 is the intercept, $\mathbf{x}_i$ is the $1 \times p$ vector of covariates, $\boldsymbol{\beta}$ is the $1 \times p$ vector of coefficients, λ is the lasso penalty parameter that must be ≥ 0, and κ_j are coefficient-level weights.

When $\lambda = 0$, there is no penalty term, and Q is the objective function for a version of the reweighted least-squares estimator.

By default, the coefficient-level weights κ_j are 1. The heteroskedastic plugin estimator uses coefficient-level weights that differ from 1. In addition, they may be set to other values using option `penaltywt()`.

`sqrtlasso` uses the coordinate descent algorithm to minimize Q for a given value of λ. See Friedman et al. (2007) for an introduction to the coordinate descent algorithm.

The numerical problem is made much easier and more stable by standardizing all the covariates to have mean 0 and standard deviation 1. The standardization also removes β_0 from the problem.

The grid of values for λ is specified as described in *Methods and formulas* in [LASSO] **lasso**.

As with lasso and elastic net, we need to select a value of λ^*. The available selection methods are `selection(cv)` (CV, the default), `selection(plugin)`, `selection(bic)`, and `selection(none)`. The square-root lasso was designed to facilitate the derivation of the plugin estimator for λ^* discussed below. CV and BIC for the square-root lasso use the same algorithm as the regular lasso; see *Methods and formulas* in [LASSO] **lasso** for details.

If option `cluster()` is specified, the penalized objective function with clusters is

$$Q = \sqrt{\frac{1}{N_{\text{clust}}} \sum_{i=1}^{N_{\text{clust}}} \left\{ \frac{1}{T_i} \sum_{t=1}^{T_i} w_{it} (y_{it} - \beta_0 - \mathbf{x}_{it}\boldsymbol{\beta}')^2 \right\}} + \frac{\lambda}{N_{\text{clust}}} \sum_{j=1}^{p} \kappa_j |\beta_j|$$

where N_{clust} is the total number of clusters and T_i is the number of observations in cluster i. For the tth observation in cluster i, w_{it} is its observational level weight, y_{it} is the dependent variable, and $\mathbf{x}_{it}$ are the covariates.

Plugin estimators

The same formula for the plugin estimator is used for the homoskedastic and the heteroskedastic cases with the square-root lasso. This result is essentially why the square-root lasso was derived; see Belloni, Chernozhukov, and Wang (2011). In the homoskedastic case, the coefficient-level weights are all 1 because the variables have been normalized. In the heteroskedastic case, the coefficient-level weights are estimated using algorithm 1, which comes from Belloni, Chernozhukov, and Wang (2011, 769).

The formula for λ^* is

$$\lambda_{\text{sqrt}} = 2c\sqrt{N}\,\Phi^{-1}\left(1 - \frac{\gamma}{2p}\right)$$

where $c = 1.1$ per the recommendation of Belloni and Chernozhukov (2011), N is the sample size, γ is the probability of not removing variable x_j when it has a coefficient of 0, and p is the number of candidate covariates in the model. Also, per the recommendation of Belloni and Chernozhukov (2011), we set $\gamma = 0.1/\ln[\max\{p, N\}]$.

Algorithm 1: Estimate coefficient-level weights for the heteroskedastic case

1. Remove the mean and standardize each of the covariates x_j to have variance one. Remove the mean from y.
2. Initialize the maximum number of iterations $K = 15$, initialize the iteration counter $k = 0$, and initialize each of the coefficient-level weights,

$$\kappa_{j,0} = \max_{1\le i\le N}|x_{ij}| \text{ for } j \in \{1,\ldots,p\}$$

3. Update $k = k + 1$, and estimate the square-root lasso coefficients $\widehat{\boldsymbol{\beta}}$ using the coefficient-level weights $\kappa_{j,k-1}$ and the above formula for λ_{sqrt}.
4. Update the coefficient-level weights,

$$\kappa_{j,k} = \max\left\{1, \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{ij}r_i)^2}}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}r_i^2}}\right\}$$

where $r_i = y_i - \mathbf{x}_i\widehat{\boldsymbol{\beta}}'$.

References

Belloni, A., and V. Chernozhukov. 2011. "High dimensional sparse econometric models: An Introduction". In *Inverse Problems of High-Dimensional Estimation*, edited by P. Alguier, E. Gautier, and G. Stoltz, 121–156. Berlin: Springer. https://doi.org/10.1007/978-3-642-19989-9_3.

Belloni, A., V. Chernozhukov, and L. Wang. 2011. Square-root lasso: Pivotal recovery of sparse signals via conic programming. *Biometrika* 98: 791–806. https://doi.org/10.1093/biomet/asr043.

Friedman, J. H., T. J. Hastie, H. Höfling, and R. J. Tibshirani. 2007. Pathwise coordinate optimization. *Annals of Applied Statistics* 1: 302–332. https://doi.org/10.1214/07-AOAS131.

Hastie, T. J., R. J. Tibshirani, and M. Wainwright. 2015. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Boca Raton, FL: CRC Press. https://doi.org/10.1201/b18401.

Also see

[LASSO] **lasso postestimation** — Postestimation tools for lasso for prediction

[LASSO] **elasticnet** — Elastic net for prediction and model selection

[LASSO] **lasso** — Lasso for prediction and model selection

[R] **regress** — Linear regression

[U] 20 Estimation and postestimation commands

xpoivregress — Cross-fit partialing-out lasso instrumental-variables regression

Description

`xpoivregress` fits a lasso instrumental-variables linear regression model and reports coefficients along with standard errors, test statistics, and confidence intervals for specified covariates of interest. The covariates of interest may be endogenous or exogenous. The cross-fit partialing-out method is used to estimate effects for these variables and to select from potential control variables and instruments to be included in the model.

Quick start

Estimate a coefficient for endogenous `d1` in a linear regression of `y` on `d1`, and include `x1` to `x100` as potential control variables and `z1` to `z100` as potential instruments to be selected by lassos

```
xpoivregress y (d1 = z1-z100), controls(x1-x100)
```

Same as above, and estimate the coefficient for the exogenous `d2`

```
xpoivregress y d2 (d1 = z1-z100), controls(x1-x100)
```

Same as above, but use 20 folds instead of 10 for cross-fitting

```
xpoivregress y d2 (d1 = z1-z100), controls(x1-x100) xfolds(20)
```

Same as above, but repeat the cross-fitting procedure 15 times, and average the results

```
xpoivregress y d2 (d1 = z1-z100), controls(x1-x100) xfolds(20)  ///
   resample(15)
```

Use cross-validation (CV) instead of a plugin iterative formula to select the optimal λ^* in each lasso

```
xpoivregress y d2 (d1 = z1-z100), controls(x1-x100) selection(cv)
```

Same as above, and set a random-number seed for reproducibility

```
xpoivregress y d2 (d1 = z1-z100), controls(x1-x100) selection(cv) ///
   rseed(28)
```

Specify CV for the lasso for `y` only, with the stopping rule criterion turned off

```
xpoivregress y d2 (d1 = z1-z100), controls(x1-x100)       ///
   lasso(y, selection(cv), stop(0))
```

Same as above, but apply the option to the lassos for `y`, `d2`, and `d1`

```
xpoivregress y d2 (d1 = z1-z100), controls(x1-x100)       ///
   lasso(*, selection(cv), stop(0))
```

Menu

Statistics > Lasso > Lasso inferential models > Continuous outcomes > Cross-fit partialing-out IV model

Syntax

`xpoivregress` *depvar* [*exovars*] `(`*endovars* = *instrumvars*`)` [*if*] [*in*] `,`
 `controls(`[`(`*alwaysvars*`)`] *othervars*`)` [*options*]

Coefficients and standard errors are estimated for the exogenous variables, *exovars*, and the endogenous variables, *endovars*. The set of instrumental variables, *instrumvars*, may be high dimensional.

options	Description
Model	
* `controls(`[`(`*alwaysvars*`)`] *othervars*`)`	*alwaysvars* and *othervars* are control variables for *depvar*, *exovars*, and *endovars*; *instrumvars* are an additional set of control variables that apply only to the *endovars*; *alwaysvars* are always included; lassos choose whether to include or exclude *othervars*
`selection(plugin)`	use a plugin iterative formula to select an optimal value of the lasso penalty parameter λ^* for each lasso; the default
`selection(cv)`	use CV to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(adaptive)`	use adaptive lasso to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(bic)`	use BIC to select an optimal value of the lasso penalty parameter λ^* for each lasso
`sqrtlasso`	use square-root lassos
`xfolds(#)`	use # folds for cross-fitting
`resample`[`(#)`]	repeat sample splitting # times and average results
`technique(dml1 | dml2)`	use either double machine learning 1 (`dml1`) or double machine learning 2 (`dml2`) estimation technique; `dml2` is the default
`missingok`	after fitting lassos, ignore missing values in any *instrumvars* or *othervars* not selected, and include these observations in the final model
SE/Robust	
`vce(`*vcetype*`)`	*vcetype* may be `robust` (the default) or `cluster` *clustvar*
Reporting	
`level(#)`	set confidence level; default is `level(95)`
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
Optimization	
[`no`]`log`	display or suppress an iteration log
`verbose`	display a verbose iteration log
`rseed(#)`	set random-number seed
Advanced	
`lasso(`*varlist*`,` *lasso_options*`)`	specify options for the lassos for variables in *varlist*; may be repeated
`sqrtlasso(`*varlist*`,` *lasso_options*`)`	specify options for square-root lassos for variables in *varlist*; may be repeated

reestimate	refit the model after using lassoselect to select a different λ^*
noheader	do not display the header on the coefficient table
coeflegend	display legend instead of statistics

*controls() is required.

exovars, *endovars*, *instrumvars*, *alwaysvars*, and *othervars* may contain factor variables. Base levels of factor variables cannot be set for *instrumvars*, *alwaysvars*, and *othervars*. See **[U] 11.4.3 Factor variables**.

collect is allowed; see **[U] 11.1.10 Prefix commands**.

reestimate, noheader, and coeflegend do not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

controls([(*alwaysvars*)] *othervars*) specifies the set of control variables, which control for omitted variables. Control variables are also known as confounding variables. *alwaysvars* are variables that are always to be included in lassos. *alwaysvars* are optional. *othervars* are variables that lassos will choose to include or exclude. The instrumental variables, *instrumvars*, are an additional set of control variables, but they apply only to the *endovars*. controls() is required.

xpoivregress fits lassos for *depvar* and each one of the *exovars* and *endovars*. The control variables for the lassos for *depvar* and *exovars* are *alwaysvars* (always included) and *othervars* (lasso will include or exclude). The control variables for lassos for *endovars* are *exovars* (always included), *alwaysvars* (always included), *instrumvars* (lasso will include or exclude), and *othervars* (lasso will include or exclude).

selection(plugin | cv | adaptive | bic) specifies the selection method for choosing an optimal value of the lasso penalty parameter λ^* for each lasso or square-root lasso estimation. Separate lassos are estimated for *depvar* and each variable in *varsofinterest*. Specifying selection() changes the selection method for all of these lassos. You can specify different selection methods for different lassos using the option lasso() or sqrtlasso(). When lasso() or sqrtlasso() is used to specify a different selection method for the lassos of some variables, they override the global setting made using selection() for the specified variables.

selection(plugin) is the default. It selects λ^* based on a “plugin” iterative formula dependent on the data. See [LASSO] **lasso options**.

selection(cv) selects the λ^* that gives the minimum of the CV function. See [LASSO] **lasso options**.

selection(adaptive) selects λ^* using the adaptive lasso selection method. It cannot be specified when sqrtlasso is specified. See [LASSO] **lasso options**.

selection(bic) selects the λ^* that gives the minimum of the BIC function. See [LASSO] **lasso options**.

sqrtlasso specifies that square-root lassos be done rather than regular lassos. The option lasso() can be used with sqrtlasso to specify that regular lasso be done for some variables, overriding the global sqrtlasso setting for these variables. See [LASSO] **lasso options**.

xfolds(#) specifies the number of folds for cross-fitting. The default is xfolds(10).

resample[(#)] specifies that sample splitting be repeated and results averaged. This reduces the effects of the randomness of sample splitting on the estimated coefficients. Not specifying resample or resample(#) is equivalent to specifying resample(1). In other words, by default no resampling is done. Specifying resample alone is equivalent to specifying resample(10). That is, sample splitting is repeated 10 times. For each sample split, lassos are computed. So when this option is not specified, lassos are repeated xfolds(#) times. But when resample(#) is specified, lassos are repeated xfolds(#) $\times$ resample(#) times. Thus, while we recommend using resample to get final results, note that it can be an extremely time-consuming procedure.

technique(dml1 | dml2) specifies which cross-fitting technique is used, either double machine learning 1 (dml1) or double machine learning 2 (dml2). For both techniques, the initial estimation steps are the same. The sample is split into K = xfolds(#) folds. Then, coefficients on the controls are estimated using only the observations not in the kth fold, for $k = 1, 2, \dots, K$. Moment conditions for the coefficients on the *varsofinterest* are formed using the observations in fold k. The default technique, dml2, solves the moment conditions jointly across all the observations. The optional technique, dml1, solves the moment conditions in each fold k to produce K different estimates, which are then averaged to form a single vector of estimates. See *Methods and formulas*.

missingok specifies that, after fitting lassos, the estimation sample be redefined based on only the nonmissing observations of variables in the final model. In all cases, any observation with missing values for *depvar*, *exovars*, *endovars*, *instrumvars*, *alwaysvars*, and *othervars* is omitted from the estimation sample for the lassos. By default, the same sample is used for calculation of the coefficients of the *exovars* and *endovars* and their standard errors.

When missingok is specified, the initial estimation sample is the same as the default, but the sample used for the calculation of the coefficients of the *exovars* and *endovars* can be larger. Now observations with missing values for any *instrumvars* and *othervars* not selected will be added to the estimation sample (provided there are no missing values for any of the variables in the final model).

missingok may produce more efficient estimates when data are missing completely at random. It does, however, have the consequence that estimation samples can change when selected variables differ in models fit using different selection methods. That is, when *instrumvars* and *othervars* contain missing values, the estimation sample for a model fit using the default selection(plugin) will likely differ from the estimation sample for a model fit using, for example, selection(cv).

SE/Robust

vce(*vcetype*) specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (robust) and that allow for intragroup correlation (cluster *clustvar*); see [R] ***vce_option***.

When vce(cluster *clustvar*) is specified, all lassos also account for clustering. For each lasso, this affects how the log-likelihood function is computed and how the sample is split in cross-validation; see *Methods and formulas* in [LASSO] **lasso**. Specifying vce(cluster *clustvar*) may lead to different selected controls and therefore to different point estimates for your variable of interest when compared to the estimation that ignores clustering.

Reporting

level(#); see [R] **Estimation options**.

display_options: noci, nopvalues, noomitted, vsquish, noemptycells, baselevels, allbaselevels, nofvlabel, fvwrap(#), fvwrapon(*style*), cformat(%*fmt*), pformat(%*fmt*), sformat(%*fmt*), and nolstretch; see [R] **Estimation options**.

Optimization

[no]log displays or suppresses a log showing the progress of the estimation. By default, one-line messages indicating when each lasso estimation begins are shown. Specify verbose to see a more detailed log.

verbose displays a verbose log showing the iterations of each lasso estimation. This option is useful when doing selection(cv) or selection(adaptive). It allows you to monitor the progress of the lasso estimations for these selection methods, which can be time consuming when there are many *othervars* specified in controls() or many *instrumvars*.

rseed(#) sets the random-number seed. This option can be used to reproduce results. rseed(#) is equivalent to typing set seed # prior to running xpoivregress. Random numbers are used to produce split samples for cross-fitting. So for all selection() options, if you want to reproduce your results, you must either use this option or use set seed. See [R] **set seed**.

Advanced

lasso(*varlist*, *lasso_options*) lets you set different options for different lassos, or advanced options for all lassos. You specify a *varlist* followed by the options you want to apply to the lassos for these variables, where *varlist* consists of one or more variables from *depvar*, *exovars*, or *endovars*. _all or * may be used to specify *depvar* and all *exovars* and *endovars*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are selection(...), grid(...), stop(#), tolerance(#), dtolerance(#), and cvtolerance(#). When lasso(*varlist*, selection(...)) is specified, it overrides any global selection() option for the variables in *varlist*. It also overrides the global sqrtlasso option for these variables. See [LASSO] **lasso options**.

sqrtlasso(*varlist*, *lasso_options*) works like the option lasso(), except square-root lassos for the variables in *varlist* are done rather than regular lassos. *varlist* consists of one or more variables from *depvar*, *exovars*, or *endovars*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are selection(...), grid(...), stop(#), tolerance(#), dtolerance(#), and cvtolerance(#). When sqrtlasso(*varlist*, selection(...)) is specified, it overrides any global selection() option for the variables in *varlist*. See [LASSO] **lasso options**.

The following options are available with xpoivregress but are not shown in the dialog box:

reestimate is an advanced option that refits the xpoivregress model based on changes made to the underlying lassos using lassoselect. After running xpoivregress, you can select a different λ^* for one or more of the lassos estimated by xpoivregress. After selecting λ^*, you type xpoivregress, reestimate to refit the xpoivregress model based on the newly selected λ's.

reestimate may be combined only with reporting options.

noheader prevents the coefficient table header from being displayed.

coeflegend; see [R] **Estimation options**.

Remarks and examples

xpoivregress performs cross-fit partialing-out lasso instrumental-variables linear regression. This command estimates coefficients, standard errors, and confidence intervals and performs tests for variables of interest, both exogenous and endogenous, while using lassos to select from among potential control variables and instruments.

The instrumental-variables linear regression model is

$$y = \mathbf{d}\boldsymbol{\alpha}_d' + \mathbf{f}\boldsymbol{\alpha}_f' + \mathbf{x}\boldsymbol{\beta}' + \epsilon$$

where **d** are the endogenous variables, **f** are the exogenous variables for which we wish to make inferences, and **x** are the potential control variables from which the lassos select. In addition, lassos select from potential instrumental variables, **z**. `xpoivregress` reports estimated coefficients for $\boldsymbol{\alpha}_d$ and $\boldsymbol{\alpha}_f$. However, cross-fit partialing-out does not provide estimates of the coefficients on the control variables or their standard errors. No estimation results can be reported for $\boldsymbol{\beta}$.

For an introduction to the cross-fit partialing-out lasso method for inference, as well as the double-selection and partialing-out methods, see [LASSO] **Lasso inference intro**.

Examples that demonstrate how to use `xpoivregress` and the other lasso inference commands are presented in [LASSO] **Inference examples**. In particular, we recommend reading *1 Overview* for an introduction to the examples and to the `vl` command, which provides tools for working with the large lists of variables that are often included when using lasso methods. See *2 Fitting and interpreting inferential models* for comparisons of the different methods of fitting inferential models that are available in Stata. See *6 Fitting an inferential model with endogenous covariates* for examples and discussion specific to models that account for endogenous covariates.

If you are interested in digging deeper into the lassos that are used to select controls, see *5 Exploring inferential model lassos* in [LASSO] **Inference examples**.

Stored results

`xpoivregress` stores the following in `e()`:

Scalars	
`e(N)`	number of observations
`e(N_clust)`	number of clusters
`e(k_varsofinterest)`	number of variables of interest
`e(k_controls)`	number of potential control variables
`e(k_controls_sel)`	number of selected control variables
`e(k_inst)`	number of potential instruments
`e(k_inst_sel)`	number of selected instruments
`e(df)`	degrees of freedom for test of variables of interest
`e(chi2)`	χ^2
`e(p)`	p-value for test of variables of interest
`e(n_xfolds)`	number of folds for cross-fitting
`e(n_resample)`	number of resamples
`e(rank)`	rank of `e(V)`
Macros	
`e(cmd)`	`xpoivregress`
`e(cmdline)`	command as typed
`e(depvar)`	name of dependent variable
`e(lasso_depvars)`	names of dependent variables for all lassos
`e(varsofinterest)`	variables of interest
`e(controls)`	potential control variables
`e(controls_sel)`	selected control variables
`e(exog)`	exogenous variables
`e(endog)`	endogenous variables
`e(inst)`	potential instruments
`e(inst_sel)`	selected instruments
`e(model)`	`linear`
`e(title)`	title in estimation output

`e(clustvar)`	name of cluster variable
`e(chi2type)`	`Wald`; type of χ^2 test
`e(vce)`	*vcetype* specified in `vce()`
`e(vcetype)`	title used to label Std. err.
`e(rngstate)`	random-number state used
`e(properties)`	`b V`
`e(predict)`	program used to implement `predict`
`e(select_cmd)`	program used to implement `lassoselect`
`e(marginsnotok)`	predictions disallowed by `margins`
`e(asbalanced)`	factor variables `fvset` as `asbalanced`
`e(asobserved)`	factor variables `fvset` as `asobserved`
Matrices	
`e(b)`	coefficient vector
`e(V)`	variance–covariance matrix of the estimators
Functions	
`e(sample)`	marks estimation sample

In addition to the above, the following is stored in `r()`:

Matrices	
`r(table)`	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

`xpoivregress` implements cross-fit partialing-out lasso instrumental-variables regression. See *Methods and formulas* in [LASSO] **xporegress** for details about cross-fitting methods DML1 and DML2 and resampling the partitions. See *Methods and formulas* in [LASSO] **poivregress** for details about partialing-out lasso instrumental-variables regression. The model is

$$y = \mathbf{d}\boldsymbol{\alpha}_d' + \mathbf{f}\boldsymbol{\alpha}_f' + \mathbf{x}\boldsymbol{\beta}' + \epsilon$$

where **d** contains the J_d endogenous covariates of interest, **f** contains the J_f exogenous covariates of interest, and **x** contains the p_x controls. We also have p_z outside instrumental variables, denoted by **z**, that are correlated with **d** but not with ϵ. The number of controls in **x** and the number of instruments in **z** can be large and, in theory, can grow with the sample size; however, the number of nonzero elements in $\boldsymbol{\beta}$ and nonzero coefficients of **z** must not be too large, which is to say that the model must be sparse. See *Stata commands for inference* in [LASSO] **Lasso intro** for a discussion on what it means for the model to be sparse.

Cross-fit partialing-out lasso instrumental-variables regression algorithm

1. Randomly partition the sample into K subsamples called folds.
2. Define I_k to be the observations in fold k, and define IC_k to be the sample observations not in fold k.
3. For each $k = 1, \dots, K$, fill in the observations of $i \in I_k$ for the $J_d + J_f$ moment conditions that identify $\boldsymbol{\alpha}$. These moment conditions use out-of-sample estimates of the high-dimensional components estimated using the observations $i \in IC_k$.

a. Using the observations $i \in IC_k$, perform a linear lasso of y on $\mathbf{x}$ to select controls $\tilde{\mathbf{x}}_{k,y}$.

   This lasso can choose the lasso penalty parameter (λ^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

b. Using the observations $i \in IC_k$, fit a linear regression of y on $\tilde{\mathbf{x}}_{k,y}$, and let $\widehat{\boldsymbol{\delta}}_k$ be the estimated coefficients on $\tilde{\mathbf{x}}_{k,y}$.

c. For the observations $i \in I_k$, fill in the residual.

$$\tilde{\rho}_i = y_i - \tilde{\mathbf{x}}_{k,y,i}\widehat{\boldsymbol{\delta}}_k'$$

d. Using the observations $i \in IC_k$, for each $j = 1, \dots, J_d$, perform a linear lasso of d_j on $\mathbf{f}$, $\mathbf{x}$, and $\mathbf{z}$ to select the controls $\tilde{\mathbf{x}}_{d,k,j}$ and the instruments $\tilde{\mathbf{z}}_{k,j}$.

   Each of these lassos can choose the lasso penalty parameter (λ_j^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

e. Using the observations $i \in IC_k$, for each $j = 1, \dots, J_d$, fit a linear regression of d_j on $\mathbf{f}$, $\tilde{\mathbf{x}}_{d,k,j}$ and $\tilde{\mathbf{z}}_{k,j}$, and denote their coefficient estimates by $\widehat{\boldsymbol{\pi}}_k$, $\widehat{\boldsymbol{\gamma}}_{k,j}$, and $\widehat{\boldsymbol{\theta}}_{k,j}$.

f. For the observations $i \in I_k$, for each $j = 1, \dots, J_d$, fill in the prediction for $d_{j,i}$,

$$\widehat{d}_{j,i} = \mathbf{f}_i\widehat{\boldsymbol{\pi}}_k' + \tilde{\mathbf{x}}_{d,k,j,i}\widehat{\boldsymbol{\gamma}}_{k,j}' + \tilde{\mathbf{z}}_{k,j,i}\widehat{\boldsymbol{\theta}}_{k,j}'$$

g. Using observations $i \in IC_k$, for each $j = 1, \dots, J_d$, perform a linear lasso of $\widehat{d}_j$ on $\mathbf{x}$, and let $\check{\mathbf{x}}_j$ be the selected controls.

   Each of these lassos can choose the lasso penalty parameter (λ_j^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

h. Using observations $i \in IC_k$, for each $j = 1, \dots, J_d$, fit a linear regression of $\widehat{d}_j$ on $\check{\mathbf{x}}_j$, and let $\check{\boldsymbol{\gamma}}_j$ denote the coefficient estimates.

i. For the observations $i \in I_k$, for each $j = 1, \dots, J_d$, fill in

$$\check{d}_{j,i} = \widehat{d}_{j,i} - \check{\mathbf{x}}_{j,i}\check{\boldsymbol{\gamma}}_j'$$

j. For the observations $i \in I_k$, for each $j = 1, \dots, J_d$, fill in

$$\tilde{d}_{j,i} = d_{j,i} - \check{\mathbf{x}}_{j,i}\check{\boldsymbol{\gamma}}_j'$$

k. Using the observations $i \in IC_k$, for each $j = 1, \dots, J_f$, perform a linear lasso of f_j on $\mathbf{x}$ to select the controls $\tilde{\mathbf{x}}_{f,k,j}$.

   Each of these lassos can choose the lasso penalty parameter (λ_j^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

l. Using the observations $i \in IC_k$, for each $j = 1, \dots, J_f$, fit a linear regression of f_j on $\tilde{\mathbf{x}}_{f,k,j}$, and let $\widehat{\boldsymbol{\gamma}}_{f,k,j}$ denote the coefficient estimates.

m. For the observations $i \in I_k$, for each $j = 1, \dots, J_f$, fill in the residuals

$$\tilde{f}_{j,i} = f_{j,i} - \tilde{\mathbf{x}}_{f,k,j}\widehat{\gamma}'_{f,k,j}$$

n. For the observations $i \in I_k$, form the vector of instruments

$$\mathbf{w}_i = (\check{d}_{1,i}, \dots, \check{d}_{J_d,i}, \tilde{f}_{1,i}, \dots, \tilde{f}_{J_f,i})$$

o. For the observations $i \in I_k$, form the vector of partialed-out covariates

$$\mathbf{p}_i = (\tilde{d}_{1,i}, \dots, \tilde{d}_{J_d,i}, \tilde{f}_{1,i}, \dots, \tilde{f}_{J_f,i})$$

4. Compute the point estimates.

   For DML2, compute $\widehat{\boldsymbol{\alpha}}$ by solving the following sample-moment equations.

$$\frac{1}{n}\sum_{i=1}^{n} \mathbf{w}'_i(\tilde{\rho}_i - \mathbf{p}_i\boldsymbol{\alpha}') = \mathbf{0}$$

   For DML1, $\widehat{\boldsymbol{\alpha}}$ is given by

$$\widehat{\boldsymbol{\alpha}} = \frac{1}{K}\sum_{k=1}^{K} \widehat{\boldsymbol{\alpha}}_k$$

   where $\widehat{\boldsymbol{\alpha}}_k$ is computed by solving the sample-moment equations

$$\frac{1}{n_k}\sum_{i \in I_k} \mathbf{w}'_i(\tilde{\rho}_i - \mathbf{p}_i\boldsymbol{\alpha}'_k) = \mathbf{0}$$

   and n_k is the number of observations in I_k.

5. The VCE is estimated by

$$\widehat{\text{Var}}(\widehat{\boldsymbol{\alpha}}) = \frac{1}{n}\hat{\mathbf{J}}_0^{-1}\widehat{\boldsymbol{\Psi}}\left(\hat{\mathbf{J}}_0^{-1}\right)'$$

   where

$$\widehat{\boldsymbol{\Psi}} = \frac{1}{K}\sum_{k=1}^{K} \widehat{\boldsymbol{\Psi}}_k$$

$$\widehat{\boldsymbol{\Psi}}_k = \frac{1}{n_k}\sum_{i \in I_k} \widehat{\boldsymbol{\psi}}_i\widehat{\boldsymbol{\psi}}'_i$$

$$\widehat{\boldsymbol{\psi}}_i = \mathbf{w}'_i(\tilde{\rho}_i - \mathbf{p}_i\widehat{\boldsymbol{\alpha}}')$$

$$\hat{\mathbf{J}}_0 = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{n_k}\sum_{i \in I_k} \widehat{\boldsymbol{\psi}}^a_i\right)$$

   and

$$\widehat{\boldsymbol{\psi}}^a_i = \mathbf{w}'_i\mathbf{p}_i$$

See *Methods and formulas* in [LASSO] **lasso** for details on how the lassos in steps 3a, 3d, 3g, and 3k choose their penalty parameters (λ^*).

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **poivregress** — Partialing-out lasso instrumental-variables regression

[R] **ivregress** — Single-equation instrumental-variables regression

[U] 20 Estimation and postestimation commands

xpologit — Cross-fit partialing-out lasso logistic regression

Description Quick start Menu Syntax Options
Remarks and examples Stored results Methods and formulas Reference Also see

Description

`xpologit` fits a lasso logistic regression model and reports odds ratios along with standard errors, test statistics, and confidence intervals for specified covariates of interest. The cross-fit partialing-out method is used to estimate effects for these variables and to select from potential control variables to be included in the model.

Quick start

Report an odds ratio from a logistic regression of `y` on `d1`, and include `x1` to `x100` as potential control variables to be selected by lassos

```
xpologit y d1, controls(x1-x100)
```

Same as above, and estimate odds ratios for the levels of categorical `d2`

```
xpologit y d1 i.d2, controls(x1-x100)
```

Same as above, but use 20 folds instead of 10 for cross-fitting

```
xpologit y d1 i.d2, controls(x1-x100) xfolds(20)
```

Same as above, but repeat the cross-fitting procedure 15 times, and average the results

```
xpologit y d1 i.d2, controls(x1-x100) xfolds(20) resample(15)
```

Use cross-validation (CV) instead of a plugin iterative formula to select the optimal λ^* in each lasso

```
xpologit y d1 i.d2, controls(x1-x100) selection(cv)
```

Same as above, and set a random-number seed for reproducibility

```
xpologit y d1 i.d2, controls(x1-x100) selection(cv) rseed(28)
```

Specify CV for the lasso for `y` only, with the stopping rule criterion turned off

```
xpologit y d1 i.d2, controls(x1-x100) lasso(y, selection(cv), stop(0))
```

Same as above, but apply the option to the lassos for `y`, `d1`, and `i.d2`

```
xpologit y d1 i.d2, controls(x1-x100) lasso(*, selection(cv), stop(0))
```

Menu

Statistics > Lasso > Lasso inferential models > Binary outcomes > Cross-fit partialing-out logit model

Syntax

xpologit *depvar varsofinterest* [*if*] [*in*] ,
  controls([(*alwaysvars*)] *othervars*) [*options*]

varsofinterest are variables for which coefficients and their standard errors are estimated.

options	Description
Model	
* controls([(*alwaysvars*)] *othervars*)	*alwaysvars* and *othervars* make up the set of control variables; *alwaysvars* are always included; lassos choose whether to include or exclude *othervars*
selection(plugin)	use a plugin iterative formula to select an optimal value of the lasso penalty parameter λ^* for each lasso; the default
selection(cv)	use CV to select an optimal value of the lasso penalty parameter λ^* for each lasso
selection(adaptive)	use adaptive lasso to select an optimal value of the lasso penalty parameter λ^* for each lasso
selection(bic)	use BIC to select an optimal value of the lasso penalty parameter λ^* for each lasso
sqrtlasso	use square-root lassos
xfolds(#)	use # folds for cross-fitting
resample [(#)]	repeat sample splitting # times and average results
technique(dml1 | dml2)	use either double machine learning 1 (dml1) or double machine learning 2 (dml2) estimation technique; dml2 is the default
missingok	after fitting lassos, ignore missing values in any *othervars* not selected, and include these observations in the final model
offset(*varname*)	include *varname* in the lasso and model for *depvar* with its coefficient constrained to be 1
SE/Robust	
vce(*vcetype*)	*vcetype* may be robust (the default) or cluster *clustvar*
Reporting	
level(#)	set confidence level; default is level(95)
or	report odds ratios; the default
coef	report estimated coefficients
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
Optimization	
[no] log	display or suppress an iteration log
verbose	display a verbose iteration log
rseed(#)	set random-number seed

Advanced

`lasso(`*varlist*`,` *lasso_options*`)`	specify options for the lassos for variables in *varlist*; may be repeated
`sqrtlasso(`*varlist*`,` *lasso_options*`)`	specify options for square-root lassos for variables in *varlist*; may be repeated
`reestimate`	refit the model after using `lassoselect` to select a different λ^*
`noheader`	do not display the header on the coefficient table
`coeflegend`	display legend instead of statistics

*`controls()` is required.

varsofinterest, *alwaysvars*, and *othervars* may contain factor variables. Base levels of factor variables cannot be set for *alwaysvars* and *othervars*. See **[U] 11.4.3 Factor variables**.

`collect` is allowed; see **[U] 11.1.10 Prefix commands**.

`reestimate`, `noheader`, and `coeflegend` do not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

`controls(`[`(`*alwaysvars*`)`] *othervars*`)` specifies the set of control variables, which control for omitted variables. Control variables are also known as confounding variables. `xpologit` fits lassos for *depvar* and each of the *varsofinterest*. *alwaysvars* are variables that are always to be included in these lassos. *alwaysvars* are optional. *othervars* are variables that each lasso will choose to include or exclude. That is, each lasso will select a subset of *othervars* and other lassos will potentially select different subsets of *othervars*. `controls()` is required.

`selection(plugin` | `cv` | `adaptive` | `bic)` specifies the selection method for choosing an optimal value of the lasso penalty parameter λ^* for each lasso or square-root lasso estimation. Separate lassos are estimated for *depvar* and each variable in *varsofinterest*. Specifying `selection()` changes the selection method for all of these lassos. You can specify different selection methods for different lassos using the option `lasso()` or `sqrtlasso()`. When `lasso()` or `sqrtlasso()` is used to specify a different selection method for the lassos of some variables, they override the global setting made using `selection()` for the specified variables.

`selection(plugin)` is the default. It selects λ^* based on a "plugin" iterative formula dependent on the data. See [LASSO] **lasso options**.

`selection(cv)` selects the λ^* that gives the minimum of the CV function. See [LASSO] **lasso options**.

`selection(adaptive)` selects λ^* using the adaptive lasso selection method. It cannot be specified when `sqrtlasso` is specified. See [LASSO] **lasso options**.

`selection(bic)` selects the λ^* that gives the minimum of the BIC function. See [LASSO] **lasso options**.

`sqrtlasso` specifies that square-root lassos be done rather than regular lassos for the *varsofinterest*. This option does not apply to *depvar*. Square-root lassos are linear models, and the lasso for *depvar* is always a logit lasso. The option `lasso()` can be used with `sqrtlasso` to specify that regular lasso be done for some variables, overriding the global `sqrtlasso` setting for these variables. See [LASSO] **lasso options**.

`xfolds(#)` specifies the number of folds for cross-fitting. The default is `xfolds(10)`.

`resample`[`(#)`] specifies that sample splitting be repeated and results averaged. This reduces the effects of the randomness of sample splitting on the estimated coefficients. Not specifying `resample` or `resample(#)` is equivalent to specifying `resample(1)`. In other words, by default no resampling is done. Specifying `resample` alone is equivalent to specifying `resample(10)`. That is, sample splitting is repeated 10 times. For each sample split, lassos are computed. So when this option is not specified, lassos are repeated `xfolds(#)` times. But when `resample(#)` is specified, lassos are repeated `xfolds(#)` $\times$ `resample(#)` times. Thus, while we recommend using `resample` to get final results, note that it can be an extremely time-consuming procedure.

`technique(dml1 | dml2)` specifies which cross-fitting technique is used, either double machine learning 1 (`dml1`) or double machine learning 2 (`dml2`). For both techniques, the initial estimation steps are the same. The sample is split into $K =$ `xfolds(#)` folds. Then, coefficients on the controls are estimated using only the observations not in the kth fold, for $k = 1, 2, \dots, K$. Moment conditions for the coefficients on the *varsofinterest* are formed using the observations in fold k. The default technique, `dml2`, solves the moment conditions jointly across all the observations. The optional technique, `dml1`, solves the moment conditions in each fold k to produce K different estimates, which are then averaged to form a single vector of estimates. See *Methods and formulas*.

`missingok` specifies that, after fitting lassos, the estimation sample be redefined based on only the nonmissing observations of variables in the final model. In all cases, any observation with missing values for *depvar*, *varsofinterest*, *alwaysvars*, and *othervars* is omitted from the estimation sample for the lassos. By default, the same sample is used for calculation of the coefficients of the *varsofinterest* and their standard errors.

When `missingok` is specified, the initial estimation sample is the same as the default, but the sample used for the calculation of the coefficients of the *varsofinterest* can be larger. Now observations with missing values for any *othervars* not selected will be added to the estimation sample (provided there are no missing values for any of the variables in the final model).

`missingok` may produce more efficient estimates when data are missing completely at random. It does, however, have the consequence that estimation samples can change when selected variables differ in models fit using different selection methods. That is, when *othervars* contain missing values, the estimation sample for a model fit using the default `selection(plugin)` will likely differ from the estimation sample for a model fit using, for example, `selection(cv)`.

`offset(`*varname*`)` specifies that *varname* be included in the lasso and model for *depvar* with its coefficient constrained to be 1.

SE/Robust

`vce(`*vcetype*`)` specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (`robust`) and that allow for intragroup correlation (`cluster` *clustvar*); see [R] ***vce_option***.

When `vce(cluster` *clustvar*`)` is specified, all lassos also account for clustering. For each lasso, this affects how the log-likelihood function is computed and how the sample is split in cross-validation; see *Methods and formulas* in [LASSO] **lasso**. Specifying `vce(cluster` *clustvar*`)` may lead to different selected controls and therefore to different point estimates for your variable of interest when compared to the estimation that ignores clustering.

Reporting

`level(#)`; see [R] **Estimation options**.

`or` reports the estimated coefficients transformed to odds ratios, that is, e^α. Standard errors and confidence intervals are similarly transformed. `or` is the default.

`coef` reports the estimated coefficients α rather than the odds ratios (e^α). This option affects how results are displayed, not how they are estimated. `coef` may be specified at estimation or when replaying previously estimated results.

display_options: `noci`, `nopvalues`, `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `fvwrap(#)`, `fvwrapon(`*style*`)`, `cformat(%`*fmt*`)`, `pformat(%`*fmt*`)`, `sformat(%`*fmt*`)`, and `nolstretch`; see [R] **Estimation options**.

Optimization

[`no`]`log` displays or suppresses a log showing the progress of the estimation. By default, one-line messages indicating when each lasso estimation begins are shown. Specify `verbose` to see a more detailed log.

`verbose` displays a verbose log showing the iterations of each lasso estimation. This option is useful when doing `selection(cv)` or `selection(adaptive)`. It allows you to monitor the progress of the lasso estimations for these selection methods, which can be time consuming when there are many *othervars* specified in `controls()`.

`rseed(#)` sets the random-number seed. This option can be used to reproduce results. `rseed(#)` is equivalent to typing `set seed` # prior to running `xpologit`. Random numbers are used to produce split samples for cross-fitting. So for all `selection()` options, if you want to reproduce your results, you must either use this option or use `set seed`. See [R] **set seed**.

Advanced

`lasso(`*varlist*`,` *lasso_options*`)` lets you set different options for different lassos, or advanced options for all lassos. You specify a *varlist* followed by the options you want to apply to the lassos for these variables. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. `_all` or `*` may be used to specify *depvar* and all *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `lasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. It also overrides the global `sqrtlasso` option for these variables. See [LASSO] **lasso options**.

`sqrtlasso(`*varlist*`,` *lasso_options*`)` works like the option `lasso()`, except square-root lassos for the variables in *varlist* are done rather than regular lassos. *varlist* consists of one or more variables from *varsofinterest*. Square-root lassos are linear models, and this option cannot be used with *depvar*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `sqrtlasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. See [LASSO] **lasso options**.

The following options are available with `xpologit` but are not shown in the dialog box:

`reestimate` is an advanced option that refits the `xpologit` model based on changes made to the underlying lassos using `lassoselect`. After running `xpologit`, you can select a different λ^* for one or more of the lassos estimated by `xpologit`. After selecting λ^*, you type `xpologit, reestimate` to refit the `xpologit` model based on the newly selected λ's.

`reestimate` may be combined only with reporting options.

noheader prevents the coefficient table header from being displayed.

coeflegend; see [R] **Estimation options**.

Remarks and examples

xpologit performs cross-fit partialing-out lasso logistic regression. This command estimates odds ratios, standard errors, and confidence intervals and performs tests for variables of interest while using lassos to select from among potential control variables.

The logistic regression model is

$$\Pr(y = 1|\mathbf{d}, \mathbf{x}) = \frac{\exp(\mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}')}{1 + \exp(\mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}')}$$

where **d** are the variables for which we wish to make inferences and **x** are the potential control variables from which the lassos select. xpologit estimates the $\boldsymbol{\alpha}$ coefficients and reports the corresponding odds ratios, e^{α}. However, cross-fit partialing-out does not provide estimates of the coefficients on the control variables ($\boldsymbol{\beta}$) or their standard errors. No estimation results can be reported for $\boldsymbol{\beta}$.

For an introduction to the cross-fit partialing-out lasso method for inference, as well as the double-selection and partialing-out methods, see [LASSO] **Lasso inference intro**.

Examples that demonstrate how to use xpologit and the other lasso inference commands are presented in [LASSO] **Inference examples**. In particular, we recommend reading *1 Overview* for an introduction to the examples and to the vl command, which provides tools for working with the large lists of variables that are often included when using lasso methods. See *2 Fitting and interpreting inferential models* for comparisons of the different methods of fitting inferential models that are available in Stata. Everything we say there about methods of selection is applicable to both linear and nonlinear models. See *3 Fitting logit inferential models to binary outcomes. What is different?* for examples and discussion specific to logistic regression models. The primary difference from linear models involves interpreting the results.

If you are interested in digging deeper into the lassos that are used to select controls, see *5 Exploring inferential model lassos* in [LASSO] **Inference examples**.

Stored results

xpologit stores the following in e():

Scalars

e(N)	number of observations
e(N_clust)	number of clusters
e(k_varsofinterest)	number of variables of interest
e(k_controls)	number of potential control variables
e(k_controls_sel)	number of selected control variables
e(df)	degrees of freedom for test of variables of interest
e(chi2)	χ^2
e(p)	p-value for test of variables of interest
e(n_xfolds)	number of folds for cross-fitting
e(n_resample)	number of resamples
e(rank)	rank of e(V)

Macros	
`e(cmd)`	`xpologit`
`e(cmdline)`	command as typed
`e(depvar)`	name of dependent variable
`e(lasso_depvars)`	names of dependent variables for all lassos
`e(varsofinterest)`	variables of interest
`e(controls)`	potential control variables
`e(controls_sel)`	selected control variables
`e(model)`	`logit`
`e(title)`	title in estimation output
`e(offset)`	linear offset variable
`e(clustvar)`	name of cluster variable
`e(chi2type)`	`Wald`; type of χ^2 test
`e(vce)`	*vcetype* specified in `vce()`
`e(vcetype)`	title used to label Std. err.
`e(rngstate)`	random-number state used
`e(properties)`	`b V`
`e(predict)`	program used to implement `predict`
`e(select_cmd)`	program used to implement `lassoselect`
`e(marginsnotok)`	predictions disallowed by `margins`
`e(asbalanced)`	factor variables `fvset` as `asbalanced`
`e(asobserved)`	factor variables `fvset` as `asobserved`
Matrices	
`e(b)`	coefficient vector
`e(V)`	variance–covariance matrix of the estimators
Functions	
`e(sample)`	marks estimation sample

In addition to the above, the following is stored in `r()`:

Matrices	
`r(table)`	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

`xpologit` implements cross-fit partialing-out lasso logit regression (XPOLLR) as described in Chernozhukov et al. (2018), where they derived two versions of cross fitting that are known as double machine learning 1 (DML1) and double machine learning 2 (DML2). DML2 is the default method and corresponds with option `technique(dml2)`. Specify option `technique(dml1)` to get DML1 instead.

Methods DML1 and DML2 have a similar structure. Each does the following.

1. Partitions the sample into K folds.
2. Uses the postlasso estimates computed using the observations not in a specific fold to fill in the moment conditions for the observations in that fold.

DML1 solves the moment conditions using the observations in each fold to produce K different estimates and then averages these K estimates to produce the final estimate for the coefficients of interest. DML2 uses all the observations to solve the moment conditions to produce a single final estimate for the coefficients of interest.

The K folds are chosen once by default. Specify option `resample(#)` to have the K folds randomly selected # times. This resampling removes the dependence of the estimator on any specifically selected folds, at the cost of more computer time. See *Methods and formulas* in [LASSO] **xporegress** for details about resampling.

The regression model is

$$\mathbf{E}[y|\mathbf{d},\,\mathbf{x}] = G(\mathbf{d}\boldsymbol{\alpha}' + \beta_0 + \mathbf{x}\boldsymbol{\beta}')$$

where $G(a) = \exp(a)/\{1 + \exp(a)\}$, $\mathbf{d}$ contains the J covariates of interest, and $\mathbf{x}$ contains the p controls. The number of covariates in $\mathbf{d}$ must be small and fixed. The number of controls in $\mathbf{x}$ can be large and, in theory, can grow with the sample size; however, the number of nonzero elements in $\boldsymbol{\beta}$ must not be too large, which is to say that the model must be sparse.

XPOLLR algorithm

1. Randomly partition the sample into K subsamples called folds.
2. Define I_k to be the observations in fold k, and define IC_k to be the sample observations not in fold k.
3. For each $k = 1, \dots, K$, fill in the observations of $i \in I_k$ for the J moment conditions that identify $\boldsymbol{\alpha}$. These moment conditions use out-of-sample estimates of the high-dimensional components estimated using the observations $i \in IC_k$.
   a. Using the observations $i \in IC_k$, perform a logit lasso of y on $\mathbf{d}$ and $\mathbf{x}$ to select controls $\tilde{\mathbf{x}}_{k,y}$.

      This logit lasso can choose the lasso penalty parameter (λ_k^*) using the plugin estimator, adaptive lasso, or CV. The plugin value is the default.

   b. Using the observations $i \in IC_k$, fit a logit regression of y on $\mathbf{d}$ and $\tilde{\mathbf{x}}_{k,y}$, let $\widetilde{\boldsymbol{\alpha}}_k$ be the estimated coefficients on $\mathbf{d}$, and let $\tilde{\boldsymbol{\delta}}_k$ be the estimated coefficients on $\tilde{\mathbf{x}}_{k,y}$.

   c. For the observations $i \in I_k$, fill in the prediction for the high-dimensional component using the out-of-sample estimate $\tilde{\boldsymbol{\delta}}_k$.

      $$\tilde{s}_i = \tilde{\mathbf{x}}_{k,y,i}\tilde{\boldsymbol{\delta}}_k'$$

   d. Using the observations $i \in IC_k$, for $j = 1, \dots, J$, perform a linear lasso of d_j on $\mathbf{x}$ using observation-level weights

      $$w_i = G'(\mathbf{d}_i\widetilde{\boldsymbol{\alpha}}_k' + \tilde{s}_i)$$

      where $G'()$ is the derivative of $G()$, and denote the selected controls by $\tilde{\mathbf{x}}_{k,j}$.

      Each of these lassos can choose the lasso penalty parameter (λ_j^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

   e. Using the observations $i \in IC_k$, for $j = 1, \dots, J$, fit a linear regression of d_j on $\tilde{\mathbf{x}}_{k,j}$, and denote the coefficient estimates by $\widehat{\boldsymbol{\gamma}}_{k,j}$.

   f. For each observation $i \in I_k$, and for $j = 1, \dots, J$, fill in the instrument

      $$z_{j,i} = d_{j,i} - \tilde{\mathbf{x}}_{k,j,i}\widehat{\boldsymbol{\gamma}}_{k,j}'$$

   g. For each observation $i \in I_k$, collect the instruments into a vector $\mathbf{z}_i = (z_{1,i}, z_{2,i}, \dots, z_{J,i})$.

4. Compute the point estimates.

   For DML2, compute $\widehat{\boldsymbol{\alpha}}$ by solving the following sample-moment equations.

$$\frac{1}{n}\sum_{i=1}^{n}\{y_i - G(\mathbf{d}_i\boldsymbol{\alpha}' + \tilde{s}_i)\}\,\mathbf{z}_i' = \mathbf{0}$$

   For DML1, $\widehat{\boldsymbol{\alpha}}$ is given by

$$\widehat{\boldsymbol{\alpha}} = \frac{1}{K}\sum_{k=1}^{K}\widehat{\boldsymbol{\alpha}}_k$$

   where $\widehat{\boldsymbol{\alpha}}_k$ is computed by solving the sample-moment equations

$$\frac{1}{n_k}\sum_{i\in I_k}\{y_i - G(\mathbf{d}_i\boldsymbol{\alpha}_k' + \tilde{s}_i)\}\,\mathbf{z}_i' = \mathbf{0}$$

   and n_k is the number of observations in I_k.

5. The VCE is estimated by

$$\widehat{\text{Var}}(\widehat{\alpha}) = \frac{1}{n}\hat{\mathbf{J}}_0^{-1}\widehat{\boldsymbol{\Psi}}\left(\hat{\mathbf{J}}_0^{-1}\right)'$$

   where

$$\widehat{\boldsymbol{\Psi}} = \frac{1}{K}\sum_{k=1}^{K}\widehat{\boldsymbol{\Psi}}_k$$

$$\widehat{\boldsymbol{\Psi}}_k = \frac{1}{n_k}\sum_{i\in I_k}\widehat{\boldsymbol{\psi}}_i\widehat{\boldsymbol{\psi}}_i'$$

$$\widehat{\boldsymbol{\psi}}_i = \{y_i - G(\mathbf{d}_i\widehat{\boldsymbol{\alpha}}' + \tilde{s}_i)\}\,\mathbf{z}_i'$$

$$\hat{\mathbf{J}}_0 = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{n_k}\sum_{i\in I_k}\widehat{\boldsymbol{\psi}}_i^a\right)$$

   and

$$\widehat{\boldsymbol{\psi}}_i^a = \frac{\partial\widehat{\boldsymbol{\psi}}_i}{\partial\widehat{\boldsymbol{\alpha}}}$$

See *Methods and formulas* in [LASSO] **lasso** for details on how the lassos in steps 3a and 3d choose their penalty parameters (λ^*).

Reference

Chernozhukov, V., D. Chetverikov, M. Demirer, E. Duflo, C. B. Hansen, W. K. Newey, and J. M. Robins. 2018. Double/debiased machine learning for treatment and structural parameters. *Econometrics Journal* 21: C1–C68. https://doi.org/10.1111/ectj.12097.

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **dslogit** — Double-selection lasso logistic regression

[LASSO] **pologit** — Partialing-out lasso logistic regression

[R] **logit** — Logistic regression, reporting coefficients

[R] **logistic** — Logistic regression, reporting odds ratios

[U] 20 Estimation and postestimation commands

xpopoisson — Cross-fit partialing-out lasso Poisson regression

Description

`xpopoisson` fits a lasso Poisson regression model and reports incidence-rate ratios along with standard errors, test statistics, and confidence intervals for specified covariates of interest. The cross-fit partialing-out method is used to estimate effects for these variables and to select from potential control variables to be included in the model.

Quick start

Report an incidence-rate ratio from a Poisson regression of `y` on `d1`, and include `x1` to `x100` as potential control variables to be selected by lassos

```
xpopoisson y d1, controls(x1-x100)
```

Same as above, and estimate incidence-rate ratios for the levels of categorical `d2`

```
xpopoisson y d1 i.d2, controls(x1-x100)
```

Same as above, but use 20 folds instead of 10 for cross-fitting

```
xpopoisson y d1 i.d2, controls(x1-x100) xfolds(20)
```

Same as above, but repeat the cross-fitting procedure 15 times, and average the results

```
xpopoisson y d1 i.d2, controls(x1-x100) xfolds(20) resample(15)
```

Use cross-validation (CV) instead of a plugin iterative formula to select the optimal λ^* in each lasso

```
xpopoisson y d1 i.d2, controls(x1-x100) selection(cv)
```

Same as above, and set a random-number seed for reproducibility

```
xpopoisson y d1 i.d2, controls(x1-x100) selection(cv) rseed(28)
```

Specify CV for the lasso for `y` only, with the stopping rule criterion turned off

```
xpopoisson y d1 i.d2, controls(x1-x100) lasso(y, selection(cv), stop(0))
```

Same as above, but apply the option to the lassos for `y`, `d1`, and `i.d2`

```
xpopoisson y d1 i.d2, controls(x1-x100) lasso(*, selection(cv), stop(0))
```

Menu

Statistics > Lasso > Lasso inferential models > Count outcomes > Cross-fit partialing-out Poisson model

Syntax

`xpopoisson` *depvar varsofinterest* [*if*] [*in*] ,
`controls(`[`(`*alwaysvars*`)`] *othervars*`)` [*options*]

varsofinterest are variables for which coefficients and their standard errors are estimated.

options	Description
Model	
* `controls(`[`(`*alwaysvars*`)`] *othervars*`)`	*alwaysvars* and *othervars* make up the set of control variables; *alwaysvars* are always included; lassos choose whether to include or exclude *othervars*
`selection(plugin)`	use a plugin iterative formula to select an optimal value of the lasso penalty parameter λ^* for each lasso; the default
`selection(cv)`	use CV to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(adaptive)`	use adaptive lasso to select an optimal value of the lasso penalty parameter λ^* for each lasso
`selection(bic)`	use BIC to select an optimal value of the lasso penalty parameter λ^* for each lasso
`sqrtlasso`	use square-root lassos for *varsofinterest*
`xfolds(#)`	use # folds for cross-fitting
`resample`[`(#)`]	repeat sample splitting # times and average results
`technique(dml1` | `dml2)`	use either double machine learning 1 (`dml1`) or double machine learning 2 (`dml2`) estimation technique; `dml2` is the default
`missingok`	after fitting lassos, ignore missing values in any *othervars* not selected, and include these observations in the final model
`offset(`*varname*$_o$`)`	include *varname*$_o$ in the lasso and model for *depvar* with its coefficient constrained to be 1
`exposure(`*varname*$_e$`)`	include ln(*varname*$_e$) in the lasso and model for *depvar* with its coefficient constrained to be 1
SE/Robust	
`vce(`*vcetype*`)`	*vcetype* may be `robust` (the default) or `cluster` *clustvar*
Reporting	
`level(#)`	set confidence level; default is `level(95)`
`irr`	report incidence-rate ratios; the default
`coef`	report estimated coefficients
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
Optimization	
[`no`]`log`	display or suppress an iteration log
`verbose`	display a verbose iteration log
`rseed(#)`	set random-number seed

Advanced

`lasso(`*varlist*`,` *lasso_options*`)`	specify options for the lassos for variables in *varlist*; may be repeated
`sqrtlasso(`*varlist*`,` *lasso_options*`)`	specify options for square-root lassos for variables in *varlist*; may be repeated
`reestimate`	refit the model after using `lassoselect` to select a different λ^*
`noheader`	do not display the header on the coefficient table
`coeflegend`	display legend instead of statistics

*`controls()` is required.

varsofinterest, *alwaysvars*, and *othervars* may contain factor variables. Base levels of factor variables cannot be set for *alwaysvars* and *othervars*. See **[U] 11.4.3 Factor variables**.

`collect` is allowed; see **[U] 11.1.10 Prefix commands**.

`reestimate`, `noheader`, and `coeflegend` do not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

`controls(`[`(`*alwaysvars*`)`] *othervars*`)` specifies the set of control variables, which control for omitted variables. Control variables are also known as confounding variables. `xpopoisson` fits lassos for *depvar* and each of the *varsofinterest*. *alwaysvars* are variables that are always to be included in these lassos. *alwaysvars* are optional. *othervars* are variables that each lasso will choose to include or exclude. That is, each lasso will select a subset of *othervars* and other lassos will potentially select different subsets of *othervars*. `controls()` is required.

`selection(plugin | cv | adaptive | bic)` specifies the selection method for choosing an optimal value of the lasso penalty parameter λ^* for each lasso or square-root lasso estimation. Separate lassos are estimated for *depvar* and each variable in *varsofinterest*. Specifying `selection()` changes the selection method for all of these lassos. You can specify different selection methods for different lassos using the option `lasso()` or `sqrtlasso()`. When `lasso()` or `sqrtlasso()` is used to specify a different selection method for the lassos of some variables, they override the global setting made using `selection()` for the specified variables.

`selection(plugin)` is the default. It selects λ^* based on a "plugin" iterative formula dependent on the data. See [LASSO] **lasso options**.

`selection(cv)` selects the λ^* that gives the minimum of the CV function. See [LASSO] **lasso options**.

`selection(adaptive)` selects λ^* using the adaptive lasso selection method. It cannot be specified when `sqrtlasso` is specified. See [LASSO] **lasso options**.

`selection(bic)` selects the λ^* that gives the minimum of the BIC function. See [LASSO] **lasso options**.

`sqrtlasso` specifies that square-root lassos be done rather than regular lassos for the *varsofinterest*. This option does not apply to *depvar*. Square-root lassos are linear models, and the lasso for *depvar* is always a Poisson lasso. The option `lasso()` can be used with `sqrtlasso` to specify that regular lasso be done for some variables, overriding the global `sqrtlasso` setting for these variables. See [LASSO] **lasso options**.

`xfolds(#)` specifies the number of folds for cross-fitting. The default is `xfolds(10)`.

resample[(#)] specifies that sample splitting be repeated and results averaged. This reduces the effects of the randomness of sample splitting on the estimated coefficients. Not specifying resample or resample(#) is equivalent to specifying resample(1). In other words, by default no resampling is done. Specifying resample alone is equivalent to specifying resample(10). That is, sample splitting is repeated 10 times. For each sample split, lassos are computed. So when this option is not specified, lassos are repeated xfolds(#) times. But when resample(#) is specified, lassos are repeated xfolds(#) $\times$ resample(#) times. Thus, while we recommend using resample to get final results, note that it can be an extremely time-consuming procedure.

technique(dml1 | dml2) specifies which cross-fitting technique is used, either double machine learning 1 (dml1) or double machine learning 2 (dml2). For both techniques, the initial estimation steps are the same. The sample is split into $K =$ xfolds(#) folds. Then, coefficients on the controls are estimated using only the observations not in the kth fold, for $k = 1, 2, \dots, K$. Moment conditions for the coefficients on the *varsofinterest* are formed using the observations in fold k. The default technique, dml2, solves the moment conditions jointly across all the observations. The optional technique, dml1, solves the moment conditions in each fold k to produce K different estimates, which are then averaged to form a single vector of estimates. See *Methods and formulas*.

missingok specifies that, after fitting lassos, the estimation sample be redefined based on only the nonmissing observations of variables in the final model. In all cases, any observation with missing values for *depvar*, *varsofinterest*, *alwaysvars*, and *othervars* is omitted from the estimation sample for the lassos. By default, the same sample is used for calculation of the coefficients of the *varsofinterest* and their standard errors.

When missingok is specified, the initial estimation sample is the same as the default, but the sample used for the calculation of the coefficients of the *varsofinterest* can be larger. Now observations with missing values for any *othervars* not selected will be added to the estimation sample (provided there are no missing values for any of the variables in the final model).

missingok may produce more efficient estimates when data are missing completely at random. It does, however, have the consequence that estimation samples can change when selected variables differ in models fit using different selection methods. That is, when *othervars* contain missing values, the estimation sample for a model fit using the default selection(plugin) will likely differ from the estimation sample for a model fit using, for example, selection(cv).

offset(*varname*$_o$) specifies that *varname*$_o$ be included in the lasso and model for *depvar* with its coefficient constrained to be 1.

exposure(*varname*$_e$) specifies that ln(*varname*$_e$) be included in the lasso and model for *depvar* with its coefficient constrained to be 1.

SE/Robust

vce(*vcetype*) specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (robust) and that allow for intragroup correlation (cluster *clustvar*); see [R] ***vce_option***.

When vce(cluster *clustvar*) is specified, all lassos also account for clustering. For each lasso, this affects how the log-likelihood function is computed and how the sample is split in cross-validation; see *Methods and formulas* in [LASSO] **lasso**. Specifying vce(cluster *clustvar*) may lead to different selected controls and therefore to different point estimates for your variable of interest when compared to the estimation that ignores clustering.

Reporting

`level(#)`; see [R] **Estimation options**.

`irr` reports estimated coefficients transformed to incidence-rate ratios, that is, e^{α}. Standard errors and confidence intervals are similarly transformed. `irr` is the default.

`coef` reports the estimated coefficients α rather than the incidence-rate ratios, e^{α}. This option affects how results are displayed, not how they are estimated. `coef` may be specified at estimation or when replaying previously estimated results.

display_options: `noci`, `nopvalues`, `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `fvwrap(#)`, `fvwrapon(`*style*`)`, `cformat(%`*fmt*`)`, `pformat(%`*fmt*`)`, `sformat(%`*fmt*`)`, and `nolstretch`; see [R] **Estimation options**.

Optimization

[`no`]`log` displays or suppresses a log showing the progress of the estimation. By default, one-line messages indicating when each lasso estimation begins are shown. Specify `verbose` to see a more detailed log.

`verbose` displays a verbose log showing the iterations of each lasso estimation. This option is useful when doing `selection(cv)` or `selection(adaptive)`. It allows you to monitor the progress of the lasso estimations for these selection methods, which can be time consuming when there are many *othervars* specified in `controls()`.

`rseed(#)` sets the random-number seed. This option can be used to reproduce results. `rseed(#)` is equivalent to typing `set seed` # prior to running `xpopoisson`. Random numbers are used to produce split samples for cross-fitting. So for all `selection()` options, if you want to reproduce your results, you must either use this option or use `set seed`. See [R] **set seed**.

Advanced

`lasso(`*varlist*`,` *lasso_options*`)` lets you set different options for different lassos, or advanced options for all lassos. You specify a *varlist* followed by the options you want to apply to the lassos for these variables. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. `_all` or `*` may be used to specify *depvar* and all *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `lasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. It also overrides the global `sqrtlasso` option for these variables. See [LASSO] **lasso options**.

`sqrtlasso(`*varlist*`,` *lasso_options*`)` works like the option `lasso()`, except square-root lassos for the variables in *varlist* are done rather than regular lassos. *varlist* consists of one or more variables from *varsofinterest*. Square-root lassos are linear models, and this option cannot be used with *depvar*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are `selection(...)`, `grid(...)`, `stop(#)`, `tolerance(#)`, `dtolerance(#)`, and `cvtolerance(#)`. When `sqrtlasso(`*varlist*`, selection(...))` is specified, it overrides any global `selection()` option for the variables in *varlist*. See [LASSO] **lasso options**.

The following options are available with xpopoisson but are not shown in the dialog box:

reestimate is an advanced option that refits the xpopoisson model based on changes made to the underlying lassos using lassoselect. After running xpopoisson, you can select a different λ^* for one or more of the lassos estimated by xpopoisson. After selecting λ^*, you type xpopoisson, reestimate to refit the xpopoisson model based on the newly selected λ^*'s.

reestimate may be combined only with reporting options.

noheader prevents the coefficient table header from being displayed.

coeflegend; see [R] **Estimation options**.

Remarks and examples

xpopoisson performs cross-fit partialing-out lasso Poisson regression. This command estimates incidence-rate ratios, standard errors, and confidence intervals and performs tests for variables of interest while using lassos to select from among potential control variables.

The Poisson regression model is

$$\mathbf{E}[y|\mathbf{d},\, \mathbf{x}] = \exp(\mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}')$$

where **d** are the variables for which we wish to make inferences and **x** are the potential control variables from which the lassos select. xpopoisson estimates the $\boldsymbol{\alpha}$ coefficients and reports the corresponding incidence-rate ratios, e^{α}. However, cross-fit partialing-out does not provide estimates of the coefficients on the control variables ($\boldsymbol{\beta}$) or their standard errors. No estimation results can be reported for $\boldsymbol{\beta}$.

For an introduction to the cross-fit partialing-out lasso method for inference, as well as the double-selection and partialing-out methods, see [LASSO] **Lasso inference intro**.

Examples that demonstrate how to use xpopoisson and the other lasso inference commands are presented in [LASSO] **Inference examples**. In particular, we recommend reading *1 Overview* for an introduction to the examples and to the vl command, which provides tools for working with the large lists of variables that are often included when using lasso methods. See *2 Fitting and interpreting inferential models* for comparisons of the different methods of fitting inferential models that are available in Stata. Everything we say there about methods of selection is applicable to both linear and nonlinear models. See *4 Fitting inferential models to count outcomes. What is different?* for examples and discussion specific to Poisson regression models. The primary difference from linear models involves interpreting the results.

If you are interested in digging deeper into the lassos that are used to select controls, see *5 Exploring inferential model lassos* in [LASSO] **Inference examples**.

Stored results

xpopoisson stores the following in e():

Scalars

e(N)	number of observations
e(N_clust)	number of clusters
e(k_varsofinterest)	number of variables of interest
e(k_controls)	number of potential control variables
e(k_controls_sel)	number of selected control variables
e(df)	degrees of freedom for test of variables of interest

`e(chi2)`	χ^2
`e(p)`	p-value for test of variables of interest
`e(n_xfolds)`	number of folds for cross-fitting
`e(n_resample)`	number of resamples
`e(rank)`	rank of `e(V)`

Macros

`e(cmd)`	`xpopoisson`
`e(cmdline)`	command as typed
`e(depvar)`	name of dependent variable
`e(lasso_depvars)`	names of dependent variables for all lassos
`e(varsofinterest)`	variables of interest
`e(controls)`	potential control variables
`e(controls_sel)`	selected control variables
`e(model)`	`poisson`
`e(title)`	title in estimation output
`e(offset)`	linear offset variable
`e(clustvar)`	name of cluster variable
`e(chi2type)`	`Wald`; type of χ^2 test
`e(vce)`	*vcetype* specified in `vce()`
`e(vcetype)`	title used to label Std. err.
`e(rngstate)`	random-number state used
`e(properties)`	`b V`
`e(predict)`	program used to implement `predict`
`e(select_cmd)`	program used to implement `lassoselect`
`e(marginsnotok)`	predictions disallowed by `margins`
`e(asbalanced)`	factor variables `fvset` as `asbalanced`
`e(asobserved)`	factor variables `fvset` as `asobserved`

Matrices

`e(b)`	coefficient vector
`e(V)`	variance–covariance matrix of the estimators

Functions

`e(sample)`	marks estimation sample

In addition to the above, the following is stored in `r()`:

Matrices

`r(table)`	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

`xpopoisson` implements cross-fit partialing-out lasso Poisson regression (XPOLPR) as described in Chernozhukov et al. (2018), where they derived two versions of cross-fitting that are known as double machine learning 1 (DML1) and double machine learning 2 (DML2). DML2 is the default method and corresponds with option `technique(dml2)`. Specify option `technique(dml1)` to get DML1 instead.

Methods DML1 and DML2 have a similar structure. Each does the following:

1. Partitions the sample into K folds.
2. Uses the postlasso estimates computed using the observations not in a specific fold to fill in the moment conditions for the observations in that fold.

DML1 solves the moment conditions using the observations in each fold to produce K different estimates and then averages these K estimates to produce the final estimate for the coefficients of interest. DML2 uses all the observations to solve the moment conditions to produce a single final estimate for the coefficients of interest.

The K folds are chosen once by default. Specify option `resample(#)` to have the K folds randomly selected # times. This resampling removes the dependence of the estimator on any specifically selected folds, at the cost of more computer time. See *Methods and formulas* in [LASSO] **xporegress** for details about resampling.

The regression model is

$$\mathbf{E}[y|\mathbf{d},\, \mathbf{x}] = G(\mathbf{d}\boldsymbol{\alpha}' + \beta_0 + \mathbf{x}\boldsymbol{\beta}')$$

where $G(a) = \exp(a)$, **d** contains the J covariates of interest, and **x** contains the p controls. The number of covariates in **d** must be small and fixed. The number of controls in **x** can be large and, in theory, can grow with the sample size; however, the number of nonzero elements in $\boldsymbol{\beta}$ must not be too large, which is to say that the model must be sparse.

XPOLPR algorithm

1. Randomly partition the sample into K subsamples called folds.
2. Define I_k to be the observations in fold k, and define IC_k to be the sample observations not in fold k.
3. For each $k = 1, \ldots, K$, fill in the observations of $i \in I_k$ for the J moment conditions that identify $\boldsymbol{\alpha}$. These moment conditions use out-of-sample estimates of the high-dimensional components estimated using the observations $i \in IC_k$.
   a. Using the observations $i \in IC_k$, perform a Poisson lasso of y on **d** and **x** to select controls $\tilde{\mathbf{x}}_{k,y}$.

      This Poisson lasso can choose the lasso penalty parameter (λ_k^*) using the plugin estimator, adaptive lasso, or CV. The plugin value is the default.
   b. Using the observations $i \in IC_k$, fit a Poisson regression of y on **d** and $\tilde{\mathbf{x}}_{k,y}$, let $\widetilde{\boldsymbol{\alpha}}_k$ be the estimated coefficients on **d**, and let $\tilde{\boldsymbol{\delta}}_k$ be the estimated coefficients on $\tilde{\mathbf{x}}_{k,y}$.
   c. For the observations $i \in I_k$, fill in the prediction for the high-dimensional component using the out-of-sample estimate $\tilde{\boldsymbol{\delta}}_k$.

      $$\tilde{s}_i = \tilde{\mathbf{x}}_{k,y,i}\tilde{\boldsymbol{\delta}}_k'$$
   d. Using the observations $i \in IC_k$, for $j = 1, \ldots, J$, perform a linear lasso of d_j on **x** using observation-level weights

      $$w_i = G'(\mathbf{d}_i\widetilde{\boldsymbol{\alpha}}_k' + \tilde{s}_i)$$

      where $G'(\cdot)$ is the derivative of $G(\cdot)$, and denote the selected controls by $\tilde{\mathbf{x}}_{k,j}$.

      Each of these lassos can choose the lasso penalty parameter (λ_j^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.
   e. Using the observations $i \in IC_k$, for $j = 1, \ldots, J$, fit a linear regression of d_j on $\tilde{\mathbf{x}}_{k,j}$, and denote the coefficient estimates by $\widehat{\gamma}_{k,j}$.
   f. For each observation $i \in I_k$, and for $j = 1, \ldots, J$, fill in the instrument

      $$z_{j,i} = d_{j,i} - \tilde{\mathbf{x}}_{k,j,i}\widehat{\gamma}_{k,j}'$$

g. For each observation $i \in I_k$, collect the instruments into a vector $\mathbf{z}_i = (z_{1,i}, z_{2,i}, \dots, z_{J,i})$.

4. Compute the point estimates.

   For DML2, compute $\widehat{\boldsymbol{\alpha}}$ by solving the following sample-moment equations.

$$\frac{1}{n}\sum_{i=1}^{n} \{y_i - G(\mathbf{d}_i\boldsymbol{\alpha}' + \tilde{s}_i)\}\,\mathbf{z}_i' = \mathbf{0}$$

   For DML1, $\widehat{\boldsymbol{\alpha}}$ is given by

$$\widehat{\boldsymbol{\alpha}} = \frac{1}{K}\sum_{k=1}^{K} \widehat{\boldsymbol{\alpha}}_k$$

   where $\widehat{\boldsymbol{\alpha}}_k$ is computed by solving the sample-moment equations

$$\frac{1}{n_k}\sum_{i\in I_k} \{y_i - G(\mathbf{d}_i\boldsymbol{\alpha}_k' + \tilde{s}_i)\}\,\mathbf{z}_i' = \mathbf{0}$$

   and n_k is the number of observations in I_k.

5. The VCE is estimated by

$$\widehat{\text{Var}}(\widehat{\alpha}) = \frac{1}{n}\hat{\mathbf{J}}_0^{-1}\widehat{\boldsymbol{\Psi}}\left(\hat{\mathbf{J}}_0^{-1}\right)'$$

   where

$$\widehat{\boldsymbol{\Psi}} = \frac{1}{K}\sum_{k=1}^{K}\widehat{\boldsymbol{\Psi}}_k$$

$$\widehat{\boldsymbol{\Psi}}_k = \frac{1}{n_k}\sum_{i\in I_k}\widehat{\boldsymbol{\psi}}_i\widehat{\boldsymbol{\psi}}_i'$$

$$\widehat{\boldsymbol{\psi}}_i = \{y_i - G(\mathbf{d}_i\widehat{\boldsymbol{\alpha}}' + \tilde{s}_i)\}\,\mathbf{z}_i'$$

$$\hat{\mathbf{J}}_0 = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{n_k}\sum_{i\in I_k}\widehat{\boldsymbol{\psi}}_i^a\right)$$

   and

$$\widehat{\boldsymbol{\psi}}_i^a = \frac{\partial\widehat{\boldsymbol{\psi}}_i}{\partial\widehat{\boldsymbol{\alpha}}}$$

See *Methods and formulas* in [LASSO] **lasso** for details on how the lassos in steps 3a and 3d choose their penalty parameters (λ^*).

Reference

Chernozhukov, V., D. Chetverikov, M. Demirer, E. Duflo, C. B. Hansen, W. K. Newey, and J. M. Robins. 2018. Double/debiased machine learning for treatment and structural parameters. *Econometrics Journal* 21: C1–C68. https://doi.org/10.1111/ectj.12097.

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **dspoisson** — Double-selection lasso Poisson regression

[LASSO] **popoisson** — Partialing-out lasso Poisson regression

[R] **poisson** — Poisson regression

[U] 20 Estimation and postestimation commands

xporegress — Cross-fit partialing-out lasso linear regression

Description | Quick start | Menu | Syntax | Options
Remarks and examples | Stored results | Methods and formulas | Reference | Also see

Description

`xporegress` fits a lasso linear regression model and reports coefficients along with standard errors, test statistics, and confidence intervals for specified covariates of interest. The cross-fit partialing-out method is used to estimate effects for these variables and to select from potential control variables to be included in the model.

Quick start

Estimate a coefficient for `d1` in a linear regression of `y` on `d1`, and include `x1` to `x100` as potential control variables to be selected by lassos

```
xporegress y d1, controls(x1-x100)
```

Same as above, and estimate coefficients for the levels of categorical `d2`

```
xporegress y d1 i.d2, controls(x1-x100)
```

Same as above, but use 20 folds instead of 10 for cross-fitting

```
xporegress y d1 i.d2, controls(x1-x100) xfolds(20)
```

Same as above, but repeat the cross-fitting procedure 15 times, and average the results

```
xporegress y d1 i.d2, controls(x1-x100) xfolds(20) resample(15)
```

Use cross-validation (CV) instead of a plugin iterative formula to select the optimal λ^* in each lasso

```
xporegress y d1 i.d2, controls(x1-x100) selection(cv)
```

Same as above, and set a random-number seed for reproducibility

```
xporegress y d1 i.d2, controls(x1-x100) selection(cv) rseed(28)
```

Specify CV for the lasso for `y` only, with the stopping rule criterion turned off

```
xporegress y d1 i.d2, controls(x1-x100) lasso(y, selection(cv), stop(0))
```

Same as above, but apply the option to the lassos for `y`, `d1`, and `i.d2`

```
xporegress y d1 i.d2, controls(x1-x100) lasso(*, selection(cv), stop(0))
```

Menu

Statistics > Lasso > Lasso inferential models > Continuous outcomes > Cross-fit partialing-out model

Syntax

xporegress *depvar varsofinterest* [*if*] [*in*] ,
controls([(*alwaysvars*)] *othervars*) [*options*]

varsofinterest are variables for which coefficients and their standard errors are estimated.

options	Description
Model	
* controls([(*alwaysvars*)] *othervars*)	*alwaysvars* and *othervars* make up the set of control variables; *alwaysvars* are always included; lassos choose whether to include or exclude *othervars*
selection(plugin)	use a plugin iterative formula to select an optimal value of the lasso penalty parameter λ^* for each lasso; the default
selection(cv)	use CV to select an optimal value of the lasso penalty parameter λ^* for each lasso
selection(adaptive)	use adaptive lasso to select an optimal value of the lasso penalty parameter λ^* for each lasso
selection(bic)	use BIC to select an optimal value of the lasso penalty parameter λ^* for each lasso
sqrtlasso	use square-root lassos
xfolds(#)	use # folds for cross-fitting
resample[(#)]	repeat sample splitting # times and average results
technique(dml1 | dml2)	use either double machine learning 1 (dml1) or double machine learning 2 (dml2) estimation technique; dml2 is the default
semi	use semipartialing-out lasso regression estimator
missingok	after fitting lassos, ignore missing values in any *othervars* not selected, and include these observations in the final model
SE/Robust	
vce(*vcetype*)	*vcetype* may be robust (the default) or cluster *clustvar*
Reporting	
level(#)	set confidence level; default is level(95)
display_options	control columns and column formats, row spacing, line width, display of omitted variables and base and empty cells, and factor-variable labeling
Optimization	
[no]log	display or suppress an iteration log
verbose	display a verbose iteration log
rseed(#)	set random-number seed
Advanced	
lasso(*varlist*, *lasso_options*)	specify options for the lassos for variables in *varlist*; may be repeated
sqrtlasso(*varlist*, *lasso_options*)	specify options for square-root lassos for variables in *varlist*; may be repeated

`reestimate`	refit the model after using `lassoselect` to select a different λ^*
`noheader`	do not display the header on the coefficient table
`coeflegend`	display legend instead of statistics

*`controls()` is required.

varsofinterest, *alwaysvars*, and *othervars* may contain factor variables. Base levels of factor variables cannot be set for *alwaysvars* and *othervars*. See **[U] 11.4.3 Factor variables**.

`collect` is allowed; see **[U] 11.1.10 Prefix commands**.

`reestimate`, `noheader`, and `coeflegend` do not appear in the dialog box.

See **[U] 20 Estimation and postestimation commands** for more capabilities of estimation commands.

Options

Model

`controls(`[`(`*alwaysvars*`)`] *othervars*`)` specifies the set of control variables, which control for omitted variables. Control variables are also known as confounding variables. `xporegress` fits lassos for *depvar* and each of the *varsofinterest*. *alwaysvars* are variables that are always to be included in these lassos. *alwaysvars* are optional. *othervars* are variables that each lasso will choose to include or exclude. That is, each lasso will select a subset of *othervars* and other lassos will potentially select different subsets of *othervars*. `controls()` is required.

`selection(plugin | cv | adaptive | bic)` specifies the selection method for choosing an optimal value of the lasso penalty parameter λ^* for each lasso or square-root lasso estimation. Separate lassos are estimated for *depvar* and each variable in *varsofinterest*. Specifying `selection()` changes the selection method for all of these lassos. You can specify different selection methods for different lassos using the option `lasso()` or `sqrtlasso()`. When `lasso()` or `sqrtlasso()` is used to specify a different selection method for the lassos of some variables, they override the global setting made using `selection()` for the specified variables.

`selection(plugin)` is the default. It selects λ^* based on a "plugin" iterative formula dependent on the data. See [LASSO] **lasso options**.

`selection(cv)` selects the λ^* that gives the minimum of the CV function. See [LASSO] **lasso options**.

`selection(adaptive)` selects λ^* using the adaptive lasso selection method. It cannot be specified when `sqrtlasso` is specified. See [LASSO] **lasso options**.

`selection(bic)` selects the λ^* that gives the minimum of the BIC function. See [LASSO] **lasso options**.

`sqrtlasso` specifies that square-root lassos be done rather than regular lassos. The option `lasso()` can be used with `sqrtlasso` to specify that regular lasso be done for some variables, overriding the global `sqrtlasso` setting for these variables. See [LASSO] **lasso options**.

`xfolds(`#`)` specifies the number of folds for cross-fitting. The default is `xfolds(10)`.

`resample`[`(`#`)`] specifies that sample splitting be repeated and results averaged. This reduces the effects of the randomness of sample splitting on the estimated coefficients. Not specifying `resample` or `resample(`#`)` is equivalent to specifying `resample(1)`. In other words, by default no resampling is done. Specifying `resample` alone is equivalent to specifying `resample(10)`. That is, sample splitting is repeated 10 times. For each sample split, lassos are computed. So when this option is not specified, lassos are repeated `xfolds(#)` times. But when `resample(#)` is specified, lassos are repeated `xfolds(#)` $\times$ `resample(#)` times. Thus, while we recommend using `resample` to get final results, note that it can be an extremely time-consuming procedure.

`technique(dml1 | dml2)` specifies which cross-fitting technique is used, either double machine learning 1 (`dml1`) or double machine learning 2 (`dml2`). For both techniques, the initial estimation steps are the same. The sample is split into $K =$ `xfolds(#)` folds. Then, coefficients on the controls are estimated using only the observations not in the kth fold, for $k = 1, 2, \dots, K$. Moment conditions for the coefficients on the *varsofinterest* are formed using the observations in fold k. The default technique, `dml2`, solves the moment conditions jointly across all the observations. The optional technique, `dml1`, solves the moment conditions in each fold k to produce K different estimates, which are then averaged to form a single vector of estimates. See *Methods and formulas*.

`semi` specifies that the semipartialing-out lasso regression estimator be used instead of the fully partialing-out lasso estimator, which is the default. See *Methods and formulas*.

`missingok` specifies that, after fitting lassos, the estimation sample be redefined based on only the nonmissing observations of variables in the final model. In all cases, any observation with missing values for *depvar*, *varsofinterest*, *alwaysvars*, and *othervars* is omitted from the estimation sample for the lassos. By default, the same sample is used for calculation of the coefficients of the *varsofinterest* and their standard errors.

When `missingok` is specified, the initial estimation sample is the same as the default, but the sample used for the calculation of the coefficients of the *varsofinterest* can be larger. Now observations with missing values for any *othervars* not selected will be added to the estimation sample (provided there are no missing values for any of the variables in the final model).

`missingok` may produce more efficient estimates when data are missing completely at random. It does, however, have the consequence that estimation samples can change when selected variables differ in models fit using different selection methods. That is, when *othervars* contain missing values, the estimation sample for a model fit using the default `selection(plugin)` will likely differ from the estimation sample for a model fit using, for example, `selection(cv)`.

SE/Robust

`vce(`*vcetype*`)` specifies the type of standard error reported, which includes types that are robust to some kinds of misspecification (`robust`) and that allow for intragroup correlation (`cluster` *clustvar*); see [R] ***vce_option***.

When `vce(cluster` *clustvar*`)` is specified, all lassos also account for clustering. For each lasso, this affects how the log-likelihood function is computed and how the sample is split in cross-validation; see *Methods and formulas* in [LASSO] **lasso**. Specifying `vce(cluster` *clustvar*`)` may lead to different selected controls and therefore to different point estimates for your variable of interest when compared to the estimation that ignores clustering.

Reporting

`level(#)`; see [R] **Estimation options**.

display_options: `noci`, `nopvalues`, `noomitted`, `vsquish`, `noemptycells`, `baselevels`, `allbaselevels`, `nofvlabel`, `fvwrap(#)`, `fvwrapon(`*style*`)`, `cformat(%`*fmt*`)`, `pformat(%`*fmt*`)`, `sformat(%`*fmt*`)`, and `nolstretch`; see [R] **Estimation options**.

Optimization

[no]log displays or suppresses a log showing the progress of the estimation. By default, one-line messages indicating when each lasso estimation begins are shown. Specify verbose to see a more detailed log.

verbose displays a verbose log showing the iterations of each lasso estimation. This option is useful when doing selection(cv) or selection(adaptive). It allows you to monitor the progress of the lasso estimations for these selection methods, which can be time consuming when there are many *othervars* specified in controls().

rseed(#) sets the random-number seed. This option can be used to reproduce results. rseed(#) is equivalent to typing set seed # prior to running xporegress. Random numbers are used to produce split samples for cross-fitting. So for all selection() options, if you want to reproduce your results, you must either use this option or use set seed. See [R] **set seed**.

Advanced

lasso(*varlist*, *lasso_options*) lets you set different options for different lassos, or advanced options for all lassos. You specify a *varlist* followed by the options you want to apply to the lassos for these variables. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. _all or * may be used to specify *depvar* and all *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are selection(...), grid(...), stop(#), tolerance(#), dtolerance(#), and cvtolerance(#). When lasso(*varlist*, selection(...)) is specified, it overrides any global selection() option for the variables in *varlist*. It also overrides the global sqrtlasso option for these variables. See [LASSO] **lasso options**.

sqrtlasso(*varlist*, *lasso_options*) works like the option lasso(), except square-root lassos for the variables in *varlist* are done rather than regular lassos. *varlist* consists of one or more variables from *depvar* or *varsofinterest*. This option is repeatable as long as different variables are given in each specification. *lasso_options* are selection(...), grid(...), stop(#), tolerance(#), dtolerance(#), and cvtolerance(#). When sqrtlasso(*varlist*, selection(...)) is specified, it overrides any global selection() option for the variables in *varlist*. See [LASSO] **lasso options**.

The following options are available with xporegress but are not shown in the dialog box:

reestimate is an advanced option that refits the xporegress model based on changes made to the underlying lassos using lassoselect. After running xporegress, you can select a different λ^* for one or more of the lassos estimated by xporegress. After selecting λ^*, you type xporegress, reestimate to refit the xporegress model based on the newly selected λ's.

reestimate may be combined only with reporting options.

noheader prevents the coefficient table header from being displayed.

coeflegend; see [R] **Estimation options**.

Remarks and examples

`xporegress` performs cross-fit partialing-out lasso linear regression. This command estimates coefficients, standard errors, and confidence intervals and performs tests for variables of interest while using lassos to select from among potential control variables.

The linear regression model is

$$\mathbf{E}[y|\mathbf{d}, \mathbf{x}] = \mathbf{d}\boldsymbol{\alpha}' + \mathbf{x}\boldsymbol{\beta}'$$

where **d** are the variables for which we wish to make inferences and **x** are the potential control variables from which the lassos select. `xporegress` reports estimated coefficients for $\boldsymbol{\alpha}$. However, cross-fit partialing-out does not provide estimates of the coefficients on the control variables ($\boldsymbol{\beta}$) or their standard errors. No estimation results can be reported for $\boldsymbol{\beta}$.

For an introduction to the cross-fit partialing-out lasso method for inference, as well as the double-selection and partialing-out methods, see [LASSO] **Lasso inference intro**.

Examples that demonstrate how to use `xporegress` and the other lasso inference commands are presented in [LASSO] **Inference examples**. In particular, we recommend reading *1 Overview* for an introduction to the examples and to the `vl` command, which provides tools for working with the large lists of variables that are often included when using lasso methods. See *2 Fitting and interpreting inferential models* for examples of fitting inferential lasso linear models and comparisons of the different methods available in Stata.

If you are interested in digging deeper into the lassos that are used to select controls, see *5 Exploring inferential model lassos* in [LASSO] **Inference examples**.

Stored results

`xporegress` stores the following in `e()`:

Scalars

`e(N)`	number of observations
`e(N_clust)`	number of clusters
`e(k_varsofinterest)`	number of variables of interest
`e(k_controls)`	number of potential control variables
`e(k_controls_sel)`	number of selected control variables
`e(df)`	degrees of freedom for test of variables of interest
`e(chi2)`	χ^2
`e(p)`	*p*-value for test of variables of interest
`e(n_xfolds)`	number of folds for cross-fitting
`e(n_resample)`	number of resamples
`e(rank)`	rank of `e(V)`

Macros

`e(cmd)`	`xporegress`
`e(cmdline)`	command as typed
`e(depvar)`	name of dependent variable
`e(lasso_depvars)`	names of dependent variables for all lassos
`e(varsofinterest)`	variables of interest
`e(controls)`	potential control variables
`e(controls_sel)`	selected control variables
`e(model)`	`linear`
`e(title)`	title in estimation output
`e(clustvar)`	name of cluster variable
`e(chi2type)`	`Wald`; type of χ^2 test
`e(vce)`	*vcetype* specified in `vce()`

`e(vcetype)`	title used to label Std. err.
`e(rngstate)`	random-number state used
`e(properties)`	`b V`
`e(predict)`	program used to implement `predict`
`e(select_cmd)`	program used to implement `lassoselect`
`e(marginsnotok)`	predictions disallowed by `margins`
`e(asbalanced)`	factor variables `fvset` as `asbalanced`
`e(asobserved)`	factor variables `fvset` as `asobserved`
Matrices	
`e(b)`	coefficient vector
`e(V)`	variance–covariance matrix of the estimators
Functions	
`e(sample)`	marks estimation sample

In addition to the above, the following is stored in `r()`:

Matrices	
`r(table)`	matrix containing the coefficients with their standard errors, test statistics, p-values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any r-class command is run after the estimation command.

Methods and formulas

`xporegress` implements cross-fit partialing-out lasso linear regression as described in Chernozhukov et al. (2018), where they derived two versions of cross-fitting that are known as double machine learning 1 (DML1) and double machine learning 2 (DML2). DML2 is the default method and corresponds with option `technique(dml2)`. Specify option `technique(dml1)` to get DML1 instead.

Methods DML1 and DML2 have a similar structure. Each does the following.

1. Partitions the sample into K folds.
2. Uses the postlasso estimates computed using the observations not in a specific fold to fill in the moment conditions for the observations in that fold.

DML1 solves the moment conditions using the observations in each fold to produce K different estimates and then averages these K estimates to produce the final estimate for the coefficients of interest. DML2 uses all the observations to solve the moment conditions to produce a single final estimate for the coefficients of interest.

`xporegress` implements two methods for the partialing-out lasso regression. We call the default method partialing-out lasso regression (POLR). We call the optional method, obtained by specifying option `semi`, a semipartialing-out lasso regression (SPOLR). See *Methods and formulas* in [LASSO] **poregress** for a brief literature review of POLR and SPOLR.

The regression model is

$$\mathbf{E}[y|\mathbf{d},\, \mathbf{x}] = \mathbf{d}\boldsymbol{\alpha}' + \beta_0 + \mathbf{x}\boldsymbol{\beta}'$$

where $\mathbf{d}$ contains the J covariates of interest and $\mathbf{x}$ contains the p controls. The number of covariates in $\mathbf{d}$ must be small and fixed. The number of controls in $\mathbf{x}$ can be large and, in theory, can grow with the sample size; however, the number of nonzero elements in $\boldsymbol{\beta}$ must not be too large, which is to say that the model must be sparse.

Cross-fit POLR algorithm

1. Randomly partition the sample into K subsamples called folds.
2. Define I_k to be the observations in fold k, and define IC_k to be the sample observations not in fold k.
3. For each $k = 1, \ldots, K$, fill in the observations of $i \in I_k$ for the J moment conditions that identify $\boldsymbol{\alpha}$. These moment conditions use out-of-sample estimates of the high-dimensional components estimated using the observations $i \in IC_k$.

   a. Using the observations $i \in IC_k$, perform a linear lasso of y on $\mathbf{x}$ to select covariates $\tilde{\mathbf{x}}_{k,y}$.

      This lasso can choose the lasso penalty parameter (λ^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

   b. Using the observations $i \in IC_k$, fit a linear regression of y on $\tilde{\mathbf{x}}_{k,y}$, and let $\widehat{\boldsymbol{\delta}}_k$ be the estimated coefficients on $\tilde{\mathbf{x}}_{k,y}$.

   c. For the observations $i \in I_k$, fill in the partial outcome.

$$\tilde{y}_i = y_i - \tilde{\mathbf{x}}_{k,y,i}\widehat{\boldsymbol{\delta}}_k'$$

   d. Using the observations $i \in IC_k$, for each $j = 1, \ldots, J$, perform a linear lasso of d_j on $\mathbf{x}$ to select covariates $\tilde{\mathbf{x}}_{k,j}$.

      Each of these lassos can choose the lasso penalty parameter (λ_j^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

   e. Using the observations $i \in IC_k$, for $j = 1, \ldots, J$, fit a linear regression of d_j on $\tilde{\mathbf{x}}_{k,j}$, and denote the coefficient estimates by $\widehat{\boldsymbol{\gamma}}_{k,j}$.

   f. For each observation $i \in I_k$, and for $j = 1, \ldots, J$, fill in the instrument

$$z_{j,i} = d_{j,i} - \tilde{\mathbf{x}}_{k,j,i}\widehat{\boldsymbol{\gamma}}_{k,j}'$$

   g. For each observation $i \in I_k$, collect the instruments into a vector $\mathbf{z}_i = (z_{1,i}, z_{2,i}, \ldots, z_{J,i})$.

4. Compute the point estimates.

   For DML2, compute $\widehat{\boldsymbol{\alpha}}$ by solving the following sample-moment equations.

$$\frac{1}{n}\sum_{i=1}^{n} \mathbf{z}_i'(\tilde{y}_i - \mathbf{z}_i\boldsymbol{\alpha}') = \mathbf{0}$$

   For DML1, $\widehat{\boldsymbol{\alpha}}$ is given by

$$\widehat{\boldsymbol{\alpha}} = \frac{1}{K}\sum_{k=1}^{K} \widehat{\boldsymbol{\alpha}}_k$$

   where $\widehat{\boldsymbol{\alpha}}_k$ is computed by solving the sample-moment equations

$$\frac{1}{n_k}\sum_{i\in I_k} \mathbf{z}_i'(\tilde{y}_i - \mathbf{z}_i\boldsymbol{\alpha}_k') = \mathbf{0}$$

   and n_k is the number of observations in I_k.

5. The VCE is estimated by

$$\widehat{\mathbf{Var}}(\widehat{\boldsymbol{\alpha}}) = \frac{1}{n}\hat{\mathbf{J}}_0^{-1}\widehat{\boldsymbol{\Psi}}\left(\hat{\mathbf{J}}_0^{-1}\right)'$$

   where

$$\widehat{\boldsymbol{\Psi}} = \frac{1}{K}\sum_{k=1}^{K}\widehat{\boldsymbol{\Psi}}_k$$

$$\widehat{\boldsymbol{\Psi}}_k = \frac{1}{n_k}\sum_{i\in I_k}\widehat{\boldsymbol{\psi}}_i\widehat{\boldsymbol{\psi}}_i'$$

$$\widehat{\boldsymbol{\psi}}_i = \mathbf{z}_i'(\tilde{y}_i - \mathbf{z}_i\widehat{\boldsymbol{\alpha}}')$$

$$\hat{\mathbf{J}}_0 = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{n_k}\sum_{i\in I_k}\widehat{\boldsymbol{\psi}}_i^a\right)$$

   and

$$\widehat{\boldsymbol{\psi}}_i^a = \mathbf{z}_i'\mathbf{z}_i$$

Cross-fit SPOLR algorithm

1. Randomly partition the sample into K subsamples called folds.
2. Define I_k to be the observations in fold k, and define IC_k to be the sample observations not in fold k.
3. For each $k = 1, \ldots, K$, fill in the observations of $i \in I_k$ for the J moment conditions that identify $\boldsymbol{\alpha}$. These moment conditions use out-of-sample estimates of the high-dimensional components estimated using the observations $i \in IC_k$.
   a. Using the observations $i \in IC_k$, perform a linear lasso of y on $\mathbf{d}$ and $\mathbf{x}$ to select covariates $\tilde{\mathbf{x}}_{k,y}$.

      This lasso can choose the lasso penalty parameter (λ^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

   b. Using the observations $i \in IC_k$, fit a linear regression of y on $\mathbf{d}$ and $\tilde{\mathbf{x}}_{k,y}$, and let $\hat{\boldsymbol{\delta}}_k$ be the estimated coefficients on $\tilde{\mathbf{x}}_{k,y}$.
   c. For the observations $i \in I_k$, fill in the partial outcome.

$$\tilde{y}_i = y_i - \tilde{\mathbf{x}}_{k,y,i}\hat{\boldsymbol{\delta}}_k'$$

   d. Using the observations $i \in IC_k$, for each $j = 1, \ldots, J$, perform a linear lasso of d_j on $\mathbf{x}$ to select covariates $\tilde{\mathbf{x}}_{k,j}$.

      Each of these lassos can choose the lasso penalty parameter (λ_j^*) using one of the plugin estimators for a linear lasso, adaptive lasso, or CV. The heteroskedastic plugin estimator for the linear lasso is the default.

   e. Using the observations $i \in IC_k$, for $j = 1, \ldots, J$, fit a linear regression of d_j on $\tilde{\mathbf{x}}_{k,j}$, and denote the coefficient estimates by $\widehat{\boldsymbol{\gamma}}_{k,j}$.

f. For each observation $i \in I_k$, and for $j = 1, \ldots, J$, fill in the instrument

$$z_{j,i} = d_{j,i} - \tilde{\mathbf{x}}_{k,j,i}\widehat{\gamma}'_{k,j}$$

g. For each observation $i \in I_k$, collect the instruments into a vector $\mathbf{z}_i = (z_{1,i}, z_{2,i}, \ldots, z_{J,i})$.

4. Compute the point estimates.

   For DML2, compute $\widehat{\boldsymbol{\alpha}}$ by solving the following sample-moment equations.

$$\frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - \mathbf{d}_i\boldsymbol{\alpha}')\mathbf{z}'_i = \mathbf{0}$$

   For DML1, $\widehat{\boldsymbol{\alpha}}$ is given by

$$\widehat{\boldsymbol{\alpha}} = \frac{1}{K}\sum_{k=1}^{K}\widehat{\boldsymbol{\alpha}}_k$$

   where $\widehat{\boldsymbol{\alpha}}_k$ is computed by solving the sample-moment equations

$$\frac{1}{n_k}\sum_{i\in I_k}(\tilde{y}_i - \mathbf{d}_i\widehat{\boldsymbol{\alpha}}'_k)\mathbf{z}'_i = \mathbf{0}$$

   and n_k is the number of observations in I_k.

5. The VCE is estimated by

$$\widehat{\text{Var}}(\widehat{\boldsymbol{\alpha}}) = \frac{1}{n}\hat{\mathbf{J}}_0^{-1}\widehat{\boldsymbol{\Psi}}\left(\hat{\mathbf{J}}_0^{-1}\right)'$$

   where

$$\widehat{\boldsymbol{\Psi}} = \frac{1}{K}\sum_{k=1}^{K}\widehat{\boldsymbol{\Psi}}_k$$

$$\widehat{\boldsymbol{\Psi}}_k = \frac{1}{n_k}\sum_{i\in I_k}\widehat{\boldsymbol{\psi}}_i\widehat{\boldsymbol{\psi}}'_i$$

$$\widehat{\boldsymbol{\psi}}_i = (\tilde{y}_i - \mathbf{d}_i\widehat{\boldsymbol{\alpha}}')\mathbf{z}'_i$$

$$\hat{\mathbf{J}}_0 = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{n_k}\sum_{i\in I_k}\widehat{\boldsymbol{\psi}}^a_i\right)$$

   and

$$\boldsymbol{\psi}^a_i = \mathbf{d}_i\mathbf{z}'_i$$

See *Methods and formulas* in [LASSO] **lasso** for details on how the lassos in steps 3a and 3d of both algorithms choose their penalty parameters (λ^*).

Resampling the partitions

The K folds are chosen once by default. Specify option `resample(#)` to have the K folds randomly selected # times. This resampling removes the dependence of the estimator on any specifically selected folds, at the cost of more computer time.

Let S be the specified number of resamples.

1. For each random partition $s = 1, \dots, S$, use a cross-fit estimator to obtain the DM1 or the DM2 point estimates $\widehat{\boldsymbol{\alpha}}_s$ and the estimated VCE $\widehat{\text{Var}}(\widehat{\boldsymbol{\alpha}_s})$.
2. The mean resampling-corrected point estimates are

$$\widetilde{\boldsymbol{\alpha}} = \frac{1}{S}\sum_{s=1}^{S}\widehat{\boldsymbol{\alpha}}_s$$

3. The mean resampling-corrected estimate of the VCE is

$$\widetilde{\text{Var}}(\widetilde{\boldsymbol{\alpha}}) = \frac{1}{S}\sum_{s=1}^{S}\left\{\widehat{\text{Var}}(\widehat{\boldsymbol{\alpha}_s}) + (\widehat{\boldsymbol{\alpha}}_s - \widetilde{\boldsymbol{\alpha}})(\widehat{\boldsymbol{\alpha}}_s - \widetilde{\boldsymbol{\alpha}})'\right\}$$

Reference

Chernozhukov, V., D. Chetverikov, M. Demirer, E. Duflo, C. B. Hansen, W. K. Newey, and J. M. Robins. 2018. Double/debiased machine learning for treatment and structural parameters. *Econometrics Journal* 21: C1–C68. https://doi.org/10.1111/ectj.12097.

Also see

[LASSO] **lasso inference postestimation** — Postestimation tools for lasso inferential models

[LASSO] **dsregress** — Double-selection lasso linear regression

[LASSO] **poregress** — Partialing-out lasso linear regression

[R] **regress** — Linear regression

[U] 20 Estimation and postestimation commands

Glossary

adaptive lasso. Adaptive lasso is one of three methods that Stata provides for fitting lasso models. The other two methods are cross-validation and plugins. Adaptive lasso tends to include fewer covariates than cross-validation and more covariates than plugins. Adaptive lasso is not used for fitting square-root lasso and elastic-net models.

Bayesian information criterion, BIC. The Bayesian information criterion (BIC), also known as Schwarz criterion, is an information-based criterion used for model selection. It is given by the formula $-2 \times$ log likelihood $+ k \ln N$, where k is the number of parameters in the model and N is the sample size.

beta-min condition. Beta-min condition is a mathematical statement that the smallest nonzero coefficient, in absolute value, be sufficiently large in the true or best approximating model. The condition is seldom met in lasso models because lasso tends to omit covariates with small coefficients. That is not an issue for prediction, but it is for inference. Stata's double selection, partialing out, and cross-fit partialing out work around the issue.

coefficients of interest. See *covariates of interest and control covariates*.

control variables. See *covariates of interest and control covariates*.

covariates. Covariates, also known as explanatory variables and RHS variables, refer to the variables that appear or potentially appear on the right-hand side of a model and that predict the values of the outcome variable. This manual often refers to "potential covariates" and "selected covariates" to distinguish the variables that lasso considers from those it selects for inclusion in the model.

covariates of interest and control covariates. Covariates of interest and control covariates compose the covariates that are specified when fitting lasso models for inference. In these cases, the coefficients and standard errors for the covariates of interest are estimated and reported. The coefficients for the control covariates are not reported nor are they recoverable, but they nonetheless appear in the model to improve the measurement of the coefficients of interest.

Covariates of interest and control covariates are often called variables of interest and control variables.

The coefficients on the covariates of interest are called the coefficients of interest.

covariate selection. Covariate selection refers to processes that automatically select the covariates to be included in a model. Lasso, square-root lasso, and elastic net are three such processes. What makes them special is that they can handle so many potential covariates.

Covariate selection is handled at the same time as estimation. Covariates are included and excluded based on coefficient estimates. When estimates are 0, covariates are excluded.

cross-fitting. Cross-fitting is another term for double machine learning.

cross-validation (CV). Cross-validation (CV) is a method for fitting lasso models. The other methods that Stata provides are adaptive lasso and plugins.

The term in general refers to techniques that validate how well predictive models perform. Classic CV uses one dataset to fit the model and another to evaluate its predictions. When the term is used in connection with lasso, however, CV refers to K-fold CV, a technique that uses the same dataset to fit the model and to produce an estimate of how well the model would do if used to make out-of-sample predictions. See *folds*.

cross-validation function. The cross-validation (CV) function is calculated by first dividing the data into K folds. The model for each λ (and α for elastic net) is fit on data in all but one fold, and then the prediction on that excluded fold is computed and a measure of fit calculated. These K measures of fit are averaged to give the value of the CV function. For linear models, the CV function is the CV mean prediction error. For nonlinear models, the CV function is the CV mean deviance. CV finds the minimum of the CV function, and the value of λ (and α) that gives the minimum is the selected λ^* (and α^*).

cross-validation mean deviance. Cross-validation mean deviance is a cross-validation function that uses the observation-level deviance as a measure of fit.

cross-validation mean deviance ratio. Cross-validation mean deviance ratio is the cross-validation function using the mean of the deviance ratio as the measure of fit.

cross-validation mean prediction error. Cross-validation mean prediction error is the cross-validation function using the mean of the square of the prediction error as the measure of fit. For the linear model, the prediction error is the difference between the individual-level outcome and the linear prediction $\mathbf{x}_i'\boldsymbol{\beta}$.

data-generating process (DGP) and data-generating mechanism (DGM). Data-generating process (DGP) and data-generating mechanism (DGM) are synonyms for the underlying process that generated the data being analyzed. The scientific and statistical models that researchers fit are sometimes approximations of the DGP.

deviance. The deviance is a measure-of-fit statistic for linear and nonlinear likelihood-based models. The deviance for an observation i, D_i, is given by

$$D_i = -2(l_i - l_{\text{saturated}})$$

where l_i is the observation-level likelihood and $l_{\text{saturated}}$ is the value of the saturated likelihood.

deviance null. The deviance null is the mean of the deviance evaluated for the log likelihood of a model that only includes a constant.

deviance ratio. The deviance ratio is a measure-of-fit statistic for linear and nonlinear likelihood-based models. It is given by $D2$

$$D2 = 1 - \overline{D}/(D_{\text{null}})$$

where $\overline{D}$ is the mean of the deviance and D_{null} is the deviance null.

double machine learning (DML). Double machine learning (DML) is a method for estimating the coefficients of interest and their standard errors. When lasso is used for inference, you specify the covariates of interest and the potential control covariates. DML is a family of techniques that combine sample splitting and robust moment conditions. See *double selection, partialing out, and cross-fit partialing out*.

double selection, partialing out, and cross-fit partialing out. Double selection, partialing out, and cross-fit partialing out are three different estimation techniques for performing inference on a subset of the coefficients in a lasso model. Stata provides these techniques for linear, logit, probit, Poisson, and instrumental-variables models. Cross-fit partialing out is also known as double machine learning (DML). Also see [LASSO] **Lasso inference intro**.

ds. A shorthand that we use in this manual to refer to all the double-selection inference commands—`dsregress`, `dslogit`, and `dspoisson`.

elastic net. Elastic net is a penalized estimator designed to be less likely than lasso to exclude highly collinear covariates. Stata's `elasticnet` command fits elastic-net models using cross-validation.

excluded covariates. See *covariate selection*.

folds and K-fold cross-validation. Folds and K-fold cross-validation refer to a technique for estimating how well a model would perform in out-of-sample prediction without actually having a second dataset. The same data that were used to fit the model are then divided into K approximately equal-sized, mutually exclusive subsamples called folds. For each fold k, the model is refit on the data in the other $K-1$ folds, and that result is then used to make predictions for the values in fold k. When the process is complete for all K folds, the predictions in the combined folds are compared with actual values. The number of folds, K, is usually set to 10.

included covariates. See *covariate selection*.

inference. Inference means statistical inference or scientific inference. It involves using samples of data to infer the values of parameters in the underlying population along with measures of their likely accuracy. The likely accuracy is stated in terms of probabilities, confidence intervals, credence intervals, standard errors, and other statistical measures.

Inference can also refer to scientific inference. Scientific inference is statistical inference on a causal parameter. These parameters characterize cause-and-effect relationships. Does more education cause higher incomes, or is it simply a proxy that is associated with higher incomes because those who have it are judged to be smarter or have more drive to succeed or simply spent more time with the right people? If the interest were in simply making statistical predictions, it would not matter.

in-sample R^2. The in-sample R^2 is the R^2 evaluated at the sample where the model is fit.

knots. Knots are the values of the penalty parameters at which variables in the model change.

lambda and alpha. Lambda and alpha (λ and α) are lasso's and elastic-net's penalty parameters.

Lambda is lasso's and square-root lasso's penalty parameter. Lambda is greater than or equal to 0. When it is 0, all possible covariates are included in the model. At its largest value (which is model dependent), no covariates are included. Thus, lambda orders the models.

Alpha is elastic-net's penalty parameter. Alpha is bounded by 0 and 1, inclusive. When alpha is 0, the elastic net becomes ridge regression. When alpha is 1, the elastic net becomes lasso.

lasso. Lasso has different meanings in this glossary, depending on usage.

First, we use lasso to mean lasso, the word that started as LASSO because it was an acronym for "least absolute shrinkage and selection operator".

Second, we use lasso to mean lasso and square-root lasso, which are two different types of lasso. See *square-root lasso*.

Third, we use lasso to mean lasso, square-root lasso, and elastic net. Elastic net is yet another type of lasso that uses a different penalty function. See *elastic net*.

Lasso in the broadest sense is widely used for prediction and covariate selection.

Lasso in the narrowest sense is implemented by Stata's `lasso` command. It fits linear, logit, probit, Poisson, and Cox models. It fits them using any of four methods: cross-validation, adaptive lasso, plugins (not available for Cox models), and the Bayesian information criterion function.

Square-root lasso is implemented by Stata's `sqrtlasso` command. It fits linear models using cross-validation or plugins.

Elastic net is implemented by Stata's `elasticnet` command. It fits linear, logit, probit, Poisson, and Cox models. It uses cross-validation.

Regardless of the particular lasso used, these methods estimate coefficients on potential covariates. Covariates are included and excluded based on the estimate. When estimates are 0, covariates are excluded.

lasso selection. See *covariate selection*.

nonzero coefficients. Nonzero coefficients are the coefficients estimated for the selected covariates.

not-selected covariates. Not-selected covariates is a synonym for excluded covariates; see *covariate selection*.

outcome variable. Outcome variable, also known as dependent variable and LHS variable, refers to the variable whose values are predicted by the independent variables, which are also known as covariates and RHS variables.

out-of-sample R^2. The out-of-sample R^2 is the R^2 evaluated for a sample distinct from the one for which the model was fit.

penalized coefficients. Penalized coefficients are the coefficient estimates produced by lasso when the covariates are not standardized to have a mean of 0 and standard deviation of 1.

penalized estimators. Penalized estimators are statistical estimators that minimize a measure of fit that includes a penalty term. That term penalizes models based on their complexity. Lasso, square-root lasso, and elastic net are penalized estimators.

What distinguishes lasso from elastic net, and is the only thing that distinguishes them, is the particular form of the penalty term. Lasso uses the sum of the absolute values of the coefficients for the included covariates. Elastic net uses the same penalty term plus the sum of the squared coefficients. The additional term is designed to prevent exclusion of highly collinear covariates.

Square-root lasso uses the same penalty term as lasso, but the form of the objective function to which the penalty is added differs.

penalty loadings. Penalty loadings refer to coefficient-specific penalty weights in adaptive lasso and plugins. Allowing coefficients to have different penalty weights improves the model chosen by lasso, square-root lasso, and elastic net.

penalty parameter. Penalty parameter is the formal term for lambda (λ), lasso's and square-root lasso's penalty parameter, and alpha (α), elastic-net's penalty parameter. See *lambda and alpha*.

plugins. Plugins are the method for fitting lasso and square-root lasso models, but not elastic-net models. It is an alternative to cross-validation. Cross-validation tends to include more covariates than are justified, at least in comparison with the best approximating model. Plugins were developed to address this problem. Plugins have the added advantage of being quicker to execute, but they will sometimes miss important covariates that cross-validation will find.

po. A shorthand that we use in this manual to refer to all the partialing-out inference commands—`poregress`, `pologit`, `popoisson`, and `poivregress`.

postlasso coefficients. Postlasso coefficients, also known as postselection coefficients, are the estimated coefficients you would obtain if you refit the model selected by lasso. To be clear about it, you fit a linear model by using lasso. It selected covariates. You then refit the model on those covariates by using `regress`, `logit`, etc. Those are the postselection coefficients, and they will differ from those produced by lasso. They will differ because lasso is a shrinkage estimator, and that leads to the question: which are better for prediction?

There is no definitive answer to that question. The best answer we can give you is to use split samples and `lassogof` to evaluate both sets of predictions and choose the better one.

For your information, Stata's lasso commands—`lasso`, `sqrtlasso`, and `elasticnet`—provide both the lasso and the postselection coefficients. The lasso coefficients are stored in `e(b)`. The postselection coefficients are stored in `e(b_postselection)`. You can do in-sample and out-of-sample prediction with `predict`. `predict` by default uses the lasso coefficients. Specify option `postselection`, and it uses the postselection coefficients.

potential covariates. See *covariates*.

prediction and predictive modeling. Prediction and predictive modeling refer to predicting values of the outcome variable based on covariates. Prediction is what lasso was originally designed to do. The variables on which the predictions are based do not necessarily have a cause-and-effect relationship with the outcome. They might be proxies for the cause and effects. Also see *inference*.

regularized estimator. Regularized estimators is another term used for penalized estimators. See *penalized estimators*.

R^2. R^2 is a measure of goodness of fit. It tells you what fraction of the variance of the outcome is explained by your model.

sample splitting. Sample splitting is a way of creating two or more smaller datasets from one dataset. Observations are randomly assigned to subsamples. Stata's `splitsample` command does this. Samples are sometimes split to use the resulting subsamples in different ways. One could use the first subsample to fit the model and the second subsample to evaluate its predictions.

saturated likelihood. The saturated likelihood is the likelihood for a model that has as many estimated parameters as data points.

selected covariates. Selected covariates is synonym for included covariates; see covariate selection.

sparsity assumption. Sparsity assumption refers to a requirement for lasso to produce reliable results. That requirement is that the true model that lasso seeks has few variables, where "few" is measured relative to the number of observations in the dataset used to fit the model.

square-root lasso. Square-root lasso is a variation on lasso. Development of square-root lassos was motivated by a desire to better fit linear models with homoskedastic but not normal errors, but it can also be used with heteroskedastic errors. Stata's `sqrtlasso` command fits square-root lassos.

standardized coefficients. Standardized coefficients are the coefficient estimates produced by lasso when the covariates are standardized to have a mean of 0 and standard deviation of 1.

variable selection. See *covariate selection*.

variables of interest. See *covariates of interest and control covariates*.

xpo. A shorthand that we use in this manual to refer to all the cross-fit partialing-out inference commands—`xporegress`, `xpologit`, `xpopoisson`, and `xpoivregress`.

Subject and author index

See the combined subject index and the combined author index in the *Stata Index*.