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. `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 `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 $\lambda_{\mathrm{gmin}}/\lambda_{\mathrm{gmax}}$ 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 ($i = 1, \dots, n = \#_g$) 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(*#*).

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 stata.com

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

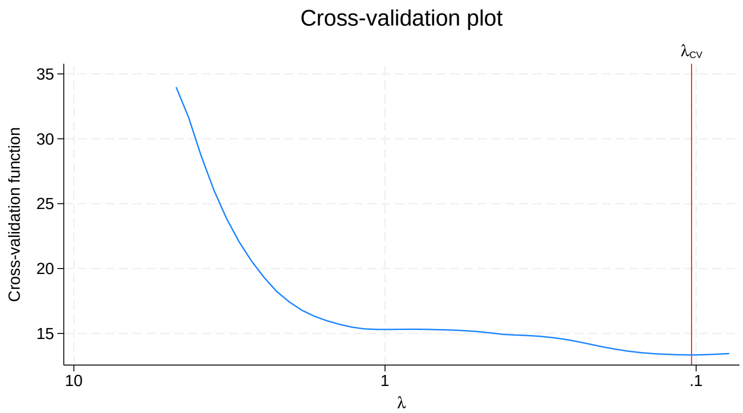
After coefficients are estimated for each λ, the value of the CV function is calculated. CV is done by dividing the data randomly into folds, by default, 10 of them. (This is the step where random numbers are used.)

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
```



λCV = .1 is the cross-validation minimum λ; # coefficients = 8.

If we want to see more values of the CV function, we can run `lasso` again using `selection(cv, alllambdas)`.

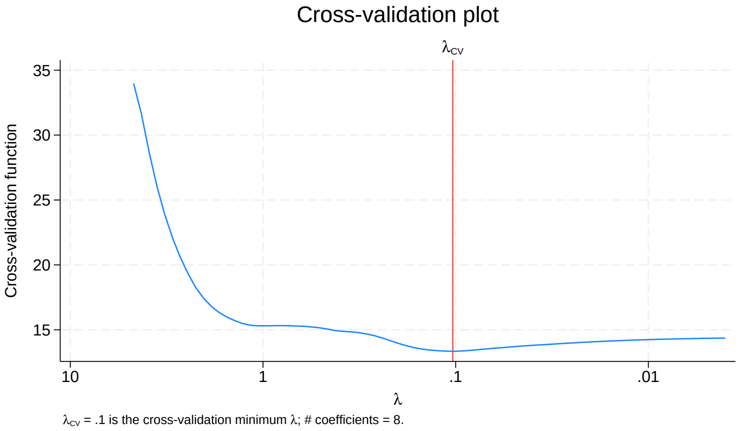
```
. set seed 1234

. 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

--------------------------------------------------------------------------
         |                                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
      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.

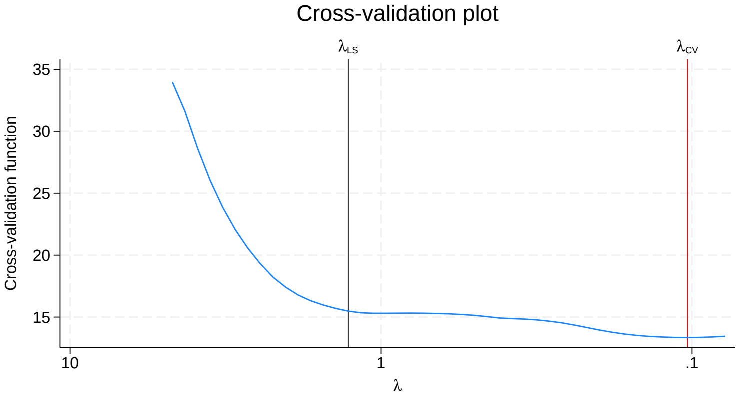
```
. lassoknots, display(cvmpe r2 osr2)

─────────────────────────────────────────────────────────────────
           │                         Out-of-
           │                CV mean   sample  In-sample
        ID │     lambda       pred.
           │                  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
```



λ_{CV} = .1 is the cross-validation minimum λ; # coefficients = 8.
λ_{LS} = 1.3 is the **lassoselect** specified λ; # coefficients = 3.

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,

$$f(\beta_0 + \mathbf{x}_i \boldsymbol{\beta}) = -y_i(\beta_0 + \mathbf{x}_i \boldsymbol{\beta}') + \ln\big\{1 + \exp(\beta_0 + \mathbf{x}_i \boldsymbol{\beta}')\big\}$$

When the model is probit,

$$f(\beta_0 + \mathbf{x}_i \boldsymbol{\beta}) = -y_i \ln\big\{\Phi(\beta_0 + \mathbf{x}_i \boldsymbol{\beta}')\big\} - (1 - y_i) \ln\big\{1 - \Phi(\beta_0 + \mathbf{x}_i \boldsymbol{\beta}')\big\}$$

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

$$Q_L = \sum_{i=1}^{N_{\text{clust}}} \left\{ \sum_{t=1}^{T_i} \widetilde{\widehat{w}}_{it} f(y_{it}, \beta_0 + \mathbf{x}_{it}\boldsymbol{\beta}') \right\} + \lambda \sum_{j=1}^{p} \kappa_j |\beta_j|$$

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, $\widetilde{\widehat{w}}_{it}$ is its normalized observational level weight, y_{it} is the dependent variable, and $\mathbf{x}_{it}$ are the covariates.

The normalized weights $\widetilde{\widehat{w}}_{it}$ are defined as

$$\widetilde{\widehat{w}}_{it} = \frac{\widehat{w}_{it}}{\sum_{i=1}^{N_{\text{clust}}} \sum_{t=1}^{T_i} \widehat{w}_{it}}$$

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

$$Q_{\text{en}} = \sum_{i=1}^{N_{\text{clust}}} \left\{ \sum_{t=1}^{T_i} \widetilde{\widehat{w}}_{it} f(y_{it}, \beta_0 + \mathbf{x}_{it}\boldsymbol{\beta}') \right\} + \lambda \sum_{j=1}^{p} \kappa_j \left\{ \frac{1-\alpha}{2} \beta_j^2 + \alpha |\beta_j| \right\}$$

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 $Q_{\rm en}$ 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

$$\widehat{\boldsymbol{\beta}}_k = g_k(y, \mathbf{x}, \widehat{\beta}_1, \ldots \widehat{\beta}_{k-1}, \widehat{\beta}_{k+1}, \ldots \widehat{\beta}_p)$$

where $g_k(y, \mathbf{x}, \widehat{\beta}_1, \ldots \widehat{\beta}_{k-1}, \widehat{\beta}_{k+1}, \ldots \widehat{\beta}_p)$ 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 $\widetilde{\boldsymbol{\beta}} = \widehat{\boldsymbol{\beta}}$.

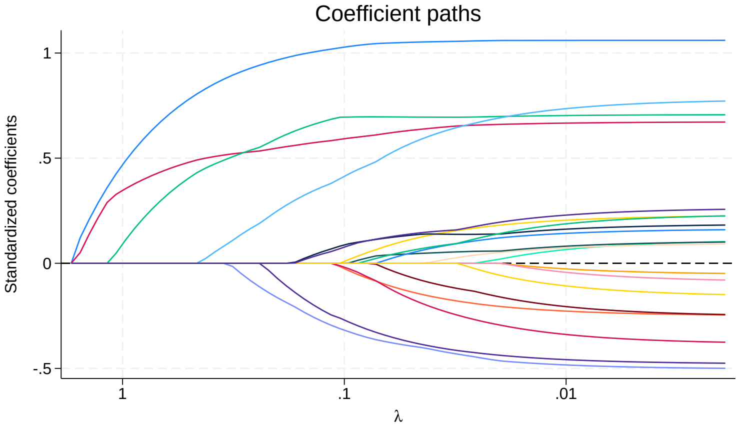
The algorithm converges when step 2 finishes and $\widehat{\boldsymbol{\beta}}$ contains the values that minimize $Q_{\rm en}$ for given values of λ and α.

When the model is `regress`, Hastie, Tibshirani, and Wainwright (2015, eq. 4.4) provide a formula for $g_k(y, \mathbf{x}, \widehat{\beta}_1, \ldots \widehat{\beta}_{k-1}, \widehat{\beta}_{k+1}, \ldots \widehat{\beta}_p)$. 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

$$\widehat{\theta} = \text{arg min}_{\theta\in\Theta}\{\text{CV}(\theta)\}$$

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, $\mathrm{Dev}\{\widehat{\theta}^{-k}(\lambda)\}$ is the deviance using the full sample and $\widehat{\theta}^{-k}(\lambda)$, and $\mathrm{Dev}^{-k}\{\widehat{\theta}^{-k}(\lambda)\}$ 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

$$P\left(\lambda^* \geq c\max\nolimits_{1\leq j\leq p}\left|\frac{1}{N\kappa_j}\sum_{i=1}^N \mathbf{h}_j(y_i, \mathbf{x}_i\boldsymbol{\beta}_0')\right|\right) \rightarrow_N 1$$

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, $\mathbf{h}_j(y_i, \widetilde{\mathbf{x}}_i\widetilde{\boldsymbol{\beta}})$ is the contribution of observation i to the unpenalized Poisson-score equation using covariates $\widetilde{\mathbf{x}}_i$ and coefficients $\widetilde{\boldsymbol{\beta}}$. In the probit case, $\mathbf{h}_j(y_i, \widetilde{\mathbf{x}}_i\widetilde{\boldsymbol{\beta}})$ is the contribution of observation i to the unpenalized probit-score equation using covariates $\widetilde{\mathbf{x}}_i$ and coefficients $\widetilde{\boldsymbol{\beta}}$.

On exit, λ contains the penalty value, and the penalty loadings are in $(\widetilde{\kappa}_1, \ldots, \widetilde{\kappa}_p)$.

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 $\widetilde{\mathbf{x}}_0$, and let $\widetilde{\mathbf{x}}_{0i}$ be the ith observation on this vector of variables.

3. Estimate the coefficients $\widetilde{\boldsymbol{\beta}}_0$ on $\widetilde{\mathbf{x}}_0$ by unpenalized QML.

4. For each $j \in \{1,\ldots,p\}$, set

$$\widetilde{\kappa}_{0,j} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\mathbf{h}_j(y_i, \widetilde{\mathbf{x}}_{0i}\widetilde{\boldsymbol{\beta}}_0)^2}$$

5. Set $k = 1$ and do the following loop. (It will be executed at most 15 times.)

   a. Using λ and loadings $\{\widetilde{\kappa}_{k-1,1}, \ldots, \widetilde{\kappa}_{k-1,p}\}$, solve the lasso to get estimates $\widetilde{\widetilde{\boldsymbol{\beta}}}_k$.

   b. Let $\widetilde{\mathbf{x}}_k$ be the covariates with nonzero coefficients in $\widetilde{\widetilde{\boldsymbol{\beta}}}_k$.

   c. Estimate the coefficients $\widetilde{\boldsymbol{\beta}}_k$ on $\widetilde{\mathbf{x}}_k$ by unpenalized QML.

   d. For each $j \in \{1,\ldots,p\}$, set

$$\widetilde{\kappa}_{k,j} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\mathbf{h}_j(y_i, \widetilde{\mathbf{x}}_{ki}\widetilde{\boldsymbol{\beta}}_k)^2}$$

   e. Set $k = k+1$.

   f. If $k > 15$ or the variables in $\widetilde{\mathbf{x}}_k$ are the same as those in $\widetilde{\mathbf{x}}_{k-1}$, set each $\widetilde{\kappa}_j = \widetilde{\kappa}_{k,j}$ 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*, ed. P. Alguier, E. Gautier, and G. Stoltz, 121–156. Berlin: Springer.

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. Unpublished manuscript. https://arxiv.org/pdf/1605.02214.pdf.

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: 1–22. 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.

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, Series A* 135: 370–384. https://doi.org/10.2307/2344614.

Quandt, R. E. 1984. Computational problems and methods. In Vol. 2 of *Handbook of Econometrics*, ed. Z. Griliches and M. D. Intriligator, 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**

For suggested citations, see the FAQ on citing Stata documentation.