Title

stata.com

lasso examples — Examples of lasso for prediction

Description Remarks and examples Also see

Description

This entry contains more examples of lasso for prediction. It assumes you have already read [LASSO] **Lasso intro** and [LASSO] **lasso**.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Overview
Using vl to manage variables
Using splitsample
Lasso linear models
Adaptive lasso
Cross-validation folds
BIC
More potential variables than observations
Factor variables in lasso
Lasso logit and probit models
Lasso Poisson models

Overview

In the examples of this entry, we use a dataset of a realistic size for lasso. It has 1,058 observations and 172 variables. Still, it is a little on the small side for lasso. Certainly, you can use lasso on datasets of this size, but lasso can also be used with datasets that have thousands or tens of thousands of variables.

The number of variables can even be greater than the number of observations. What is essential for lasso is that the set of potential variables contains a subset of variables that are in the true model (or something close to it) or are correlated with the variables in the true model.

As to how many variables there can be in the true model, we can say that the number cannot be greater than something proportional to $\sqrt{N}/\ln q$, where N is the number of observations, p is the number of potential variables, and $q = \max\{N, p\}$. We cannot, however, say what the constant of proportionality is. That this upper bound decreases with q can be viewed as the cost of performing covariate selection.

Using vl to manage variables

We will show how to use commands in the `vl` system to manage large numbers of variables. `vl` stands for "variable lists". The idea behind it is that we might want to run a lasso with hundreds or thousands or tens of thousands of variables specified as potential variables. We do not want to have to type all these variable names.

Many times, we will have a mix of different types of variables. Some we want to treat as continuous. Some we want to treat as categorical and use factor-variable operators with them to create indicator variables for their categories. See [U] **11.4.3 Factor variables**.

The first goal of the `vl` system is to help us separate variables we want to treat as categorical from those we want to treat as continuous. The second goal of the system is to help us create named variable lists we can use as arguments to `lasso` or any other Stata command simply by referring to their names.

The purpose here is to illustrate the power of `vl`, not to explain in detail how it works or show all of its features. For that, see [D] **vl**.

We load the dataset we will use in these examples.

```
. use https://www.stata-press.com/data/r17/fakesurvey
(Fictitious survey data)
```

It is simulated data designed to mimic survey data. It has 1,058 observations and 172 variables.

```
. describe
Contains data from https://www.stata-press.com/data/r17/fakesurvey.dta
 Observations:         1,058                  Fictitious survey data
    Variables:           172                  14 Jun 2020 15:31
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
id                str8    %9s                   Respondent ID
gender            byte    %8.0g      gender     Gender
age               byte    %8.0g                 Age (y)
q1                byte    %10.0g                Question 1
q2                byte    %8.0g                 Question 2
q3                byte    %8.0g      yesno      Question 3
  (output omitted)
q160              byte    %8.0g      yesno      Question 160
q161              byte    %8.0g      yesno      Question 161
check8            byte    %8.0g                 Check 8
-------------------------------------------------------------------------------
Sorted by: id
```

The variables are a mix. Some we know are integer-valued scales that we want to treat as continuous variables in our models. There are a lot of 0/1 variables, and there are some with only a few categories that we will want to turn into indicator variables. There are some with more categories that we do not yet know whether to treat as categorical or continuous.

The first `vl` subcommand we run is `vl set`. Nonnegative integer-valued variables are candidates for use as factor variables. Because factor variables cannot be negative, any variable with negative values is classified as continuous. Any variable with noninteger values is also classified as continuous.

`vl set` has two options, `categorical(#)` and `uncertain(#)`, that allow us to separate out the nonnegative integer-valued variables into three named variable lists: `vlcategorical`, `vluncertain`, and `vlcontinuous`.

When the number of levels (distinct values), L, is

$$2 \leq L \leq \texttt{categorical(\#)}$$

the variable goes in `vlcategorical`. When

$$\texttt{categorical(\#)} < L \leq \texttt{uncertain(\#)}$$

the variable goes in `vluncertain`. When

$$L > \texttt{uncertain(\#)}$$

the variable goes in `vlcontinuous`.

The defaults are `categorical(10)` and `uncertain(100)`. For our data, we do not like the defaults, so we change them. We specify `categorical(4)` and `uncertain(19)`. We also specify the option `dummy` to create a variable list, `vldummy`, consisting solely of 0/1 variables. Let's run `vl set` with these options.

```
. vl set, categorical(4) uncertain(19) dummy
```

	Macro's contents	
Macro	# Vars	Description
System		
$vldummy	99	0/1 variables
$vlcategorical	16	categorical variables
$vlcontinuous	20	continuous variables
$vluncertain	27	perhaps continuous, perhaps categorical variables
$vlother	9	all missing or constant variables

Notes

1. Review contents of **vlcategorical** and **vlcontinuous** to ensure they are correct. Type **vl list vlcategorical** and type **vl list vlcontinuous**.
2. If there are any variables in **vluncertain**, you can reallocate them to **vlcategorical**, **vlcontinuous**, or **vlother**. Type **vl list vluncertain**.
3. Use **vl move** to move variables among classifications. For example, type **vl move (x50 x80) vlcontinuous** to move variables **x50** and **x80** to the continuous classification.
4. *vlnames* are global macros. Type the *vlname* without the leading dollar sign (**$**) when using **vl** commands. Example: **vlcategorical** *not* **$vlcategorical**. Type the dollar sign with other Stata commands to get a *varlist*.

The vluncertain variable list contains all the variables we are not sure whether we want to treat as categorical or continuous. We use vl list to list the variables in vluncertain.

```
. vl list vluncertain
```

Variable	Macro	Values	Levels
q12	$vluncertain	integers >=0	5
q18	$vluncertain	integers >=0	7
q23	$vluncertain	integers >=0	10
q27	$vluncertain	integers >=0	8
q28	$vluncertain	integers >=0	15
q35	$vluncertain	integers >=0	7
q39	$vluncertain	integers >=0	5
q54	$vluncertain	integers >=0	10
q63	$vluncertain	integers >=0	7
q66	$vluncertain	integers >=0	5
q80	$vluncertain	integers >=0	5
q81	$vluncertain	integers >=0	5
q92	$vluncertain	integers >=0	5
q93	$vluncertain	integers >=0	7
q99	$vluncertain	integers >=0	5
q103	$vluncertain	integers >=0	7
q107	$vluncertain	integers >=0	18
q111	$vluncertain	integers >=0	7
q112	$vluncertain	integers >=0	7
q119	$vluncertain	integers >=0	8
q120	$vluncertain	integers >=0	7
q124	$vluncertain	integers >=0	14
q127	$vluncertain	integers >=0	5
q132	$vluncertain	integers >=0	7
q135	$vluncertain	integers >=0	10
q141	$vluncertain	integers >=0	12
q157	$vluncertain	integers >=0	7

We are going to have to go through these variables one by one and reclassify them. We know we have several seven-level Likert scales in these data. We tabulate one of them.

```
. tabulate q18
```

Question 18	Freq.	Percent	Cum.
Very strongly disagree	139	13.15	13.15
Strongly disagree	150	14.19	27.34
Disagree	146	13.81	41.15
Neither agree nor disagree	146	13.81	54.97
Agree	174	16.46	71.43
Strongly agree	146	13.81	85.24
Very strongly agree	156	14.76	100.00
Total	1,057	100.00	

We look at all the variables with seven levels, and they are all Likert scales. We want to treat them as continuous in our models, so we move them out of `vluncertain` and into `vlcontinuous`.

```
. vl move (q18 q35 q63 q93 q103 q111 q112 q120 q132 q157) vlcontinuous
note: 10 variables specified and 10 variables moved.
```

Macro	# Added/Removed
$vldummy	0
$vlcategorical	0
$vlcontinuous	10
$vluncertain	-10
$vlother	0

When variables are moved into a new `vl` system-defined variable list, they are automatically moved out of their current system-defined variable list.

In our examples, we have three variables we want to predict: `q104`, a continuous variable; `q106`, a 0/1 variable; and `q107`, a count variable. Because we are going to use the variables in `vlcategorical` and `vlcontinuous` as potential variables to select in our lassos, we do not want these dependent variables in these variable lists. We move them into `vlother`, which is intended as a place to put variables we do not want in our models.

```
. vl move (q104 q106 q107) vlother
note: 3 variables specified and 3 variables moved.
```

Macro	# Added/Removed
$vldummy	-1
$vlcategorical	0
$vlcontinuous	-1
$vluncertain	-1
$vlother	3

Notice the parentheses around the variable names when we used `vl move`. The rule for `vl` is to use parentheses around variable names and to not use parentheses for variable-list names.

The system-defined variable lists are good for a general division of variables. But we need further subdivision for our models. We have four demographic variables, which are all categorical, but we want them included in all lasso models. So we create a user-defined variable list containing these variables.

```
. vl create demographics = (gender q3 q4 q5)
note: $demographics initialized with 4 variables.
```

We want to convert the variables in `vldummy` and `vlcategorical` into indicator variables. We create a new variable list, `factors`, containing the union of these lists. Because we want to handle the variables in `demographics` separately, we remove them from `factors`.

```
. vl create factors = vldummy + vlcategorical
note: $factors initialized with 114 variables.
. vl modify factors = factors - demographics
note: 4 variables removed from $factors.
```

The `vl substitute` command allows us to apply factor-variable operators to a variable list. We turn the variables in `demographics` and `factors` into factor variables.

```
. vl substitute idemographics = i.demographics
. vl substitute ifactors = i.factors
```

We are done using vl and we save our dataset. One nice feature of vl is that the variable lists are saved with the data.

```
. label data "Fictitious survey data with vl"
. save fakesurvey_vl
file fakesurvey_vl.dta saved
```

We are now ready to run some lassos.

Using splitsample

Well, almost ready. We want to evaluate our lasso predictions on a sample that we did not use to fit the lasso. So we decide to randomly split our data into two samples of equal sizes. We will fit models on one, and we will use the other to test their predictions.

Let's load the version of our dataset that contains our variable lists. We first increase maxvar because we are going to create thousands of interactions in a later example.

```
. clear all
. set maxvar 10000
. use https://www.stata-press.com/data/r17/fakesurvey_vl
(Fictitious survey data with vl)
```

Variable lists are not automatically restored. We have to run vl rebuild to make them active.

```
. vl rebuild
Rebuilding vl macros ...
```

	Macro's contents	
Macro	# Vars	Description
System		
$vldummy	98	0/1 variables
$vlcategorical	16	categorical variables
$vlcontinuous	29	continuous variables
$vluncertain	16	perhaps continuous, perhaps categorical variables
$vlother	12	all missing or constant variables
User		
$demographics	4	variables
$factors	110	variables
$idemographics		factor-variable list
$ifactors		factor-variable list

We now use splitsample to generate a variable indicating the two subsamples.

```
. set seed 1234
. splitsample, generate(sample) nsplit(2)
. label define svalues 1 "Training" 2 "Testing"
. label values sample svalues
```

Lasso linear models

When fitting our lasso model, we can now specify variables succinctly using our `vl` variable lists. Variable lists are really global macros—we bet you already guessed this. Listing them under the header "Macro" in `vl` output was a real tip-off, right? Because they are global macros, when we use them as arguments in commands, we put a `$` in front of them.

We put parentheses around `idemographics`. This notation means that we want to force these variables into the model regardless of whether lasso wants to select them. See *Syntax* in [LASSO] **lasso**.

We also set the random-number seed using the `rseed()` option so that we can reproduce our results.

We fit `lasso` on the first subsample.

```
. lasso linear q104 ($idemographics) $ifactors $vlcontinuous
> if sample == 1, rseed(1234)

10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .9109571   no. of nonzero coef. =       4
Folds: 1...5....10   CVF = 16.93341
  (output omitted)
Grid value 23:    lambda = .1176546   no. of nonzero coef. =      74
Folds: 1...5....10   CVF = 12.17933
... cross-validation complete ... minimum found

Lasso linear model                          No. of obs        =        458
                                            No. of covariates =        273
Selection: Cross-validation                 No. of CV folds   =         10
```

ID	Description	lambda	No. of nonzero coef.	Out-of-sample R-squared	CV mean prediction error
1	first lambda	.9109571	4	0.0147	16.93341
18	lambda before	.1873395	42	0.2953	12.10991
* 19	selected lambda	.1706967	49	0.2968	12.08516
20	lambda after	.1555325	55	0.2964	12.09189
23	last lambda	.1176546	74	0.2913	12.17933

```
* lambda selected by cross-validation.

. estimates store linearcv
```

After the command finished, we used `estimates store` to store the results in memory so that we can later compare these results with those from other lassos. Note, however, that `estimates store` only saves them in memory. To save the results to disk, use

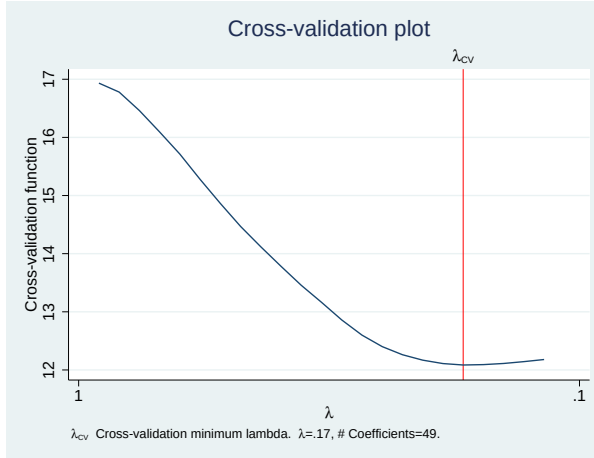
```
. estimates save filename
```

See [LASSO] **estimates store**.

The minimum of the cross-validation (CV) function was found to be at $\lambda = 0.1706967$. It selects λ^* as this λ, which corresponds to 49 variables in the model, out of 273 potential variables.

After fitting a lasso using CV to select λ, it is a good idea to plot the CV function and look at the shape of the curve around the minimum.

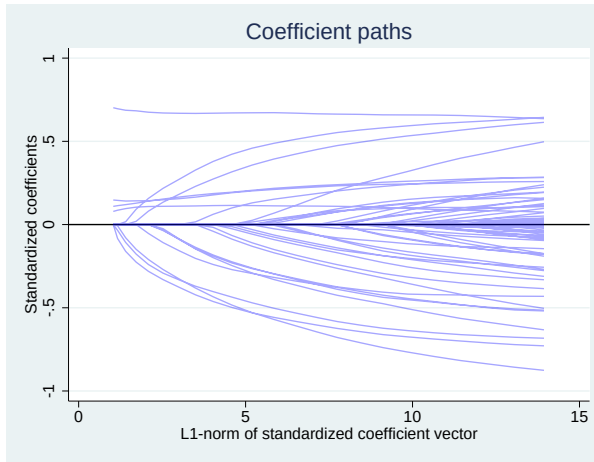
```
. cvplot
```



By default, the `lasso` command stops when it has identified a minimum. Computation time increases as λ's get smaller, so computing the CV function for smaller λ's is computationally expensive. We could specify the option `selection(cv, alllambdas)` to compute models for more small λ's. See [LASSO] **lasso** and [LASSO] **lasso fitting** for details and a description of less computationally intensive options to get more assurance that `lasso` has identified a minimum.

We can also get a plot of the size of the coefficients as they become nonzero and change as λ gets smaller. Typically, they get larger as λ gets smaller. But they can sometimes return to 0 after being nonzero.

```
. coefpath
```



We see four lines that do not start at 0. These are lines corresponding to the four variables in `idemographics` that we forced into the model.

Adaptive lasso

We are now going to run an adaptive lasso, which we do by specifying the option `selection(adaptive)`.

```
. lasso linear q104 ($idemographics) $ifactors $vlcontinuous
> if sample == 1, rseed(4321) selection(adaptive)

Lasso step 1 of 2:

10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .9109571   no. of nonzero coef. =       4
Folds: 1...5....10   CVF =   17.012
  (output omitted)
Grid value 24:    lambda = .1072025   no. of nonzero coef. =      78
Folds: 1...5....10   CVF = 12.40012
... cross-validation complete ... minimum found

Lasso step 2 of 2:

Evaluating up to 100 lambdas in grid ...
Grid value 1:     lambda = 51.68486   no. of nonzero coef. =       4
  (output omitted)
Grid value 100:   lambda = .0051685   no. of nonzero coef. =      59

10-fold cross-validation with 100 lambdas ...
Fold  1 of 10:  10....20....30....40....50....60....70....80....90....100
  (output omitted)
Fold 10 of 10:  10....20....30....40....50....60....70....80....90....100
... cross-validation complete

Lasso linear model                          No. of obs         =        458
                                            No. of covariates  =        273
Selection: Adaptive                         No. of lasso steps =          2

Final adaptive step results

------------------------------------------------------------------------------
             |                                      No. of      Out-of-   CV mean
             |                                    nonzero       sample prediction
          ID |      Description      lambda         coef.    R-squared      error
-------------+----------------------------------------------------------------
          25 |    first lambda    51.68486            4       0.0101   17.01083
          77 |   lambda before    .4095937           46       0.3985   10.33691
       * 78 | selected lambda    .3732065           46       0.3987   10.33306
          79 |    lambda after    .3400519           47       0.3985   10.33653
         124 |     last lambda    .0051685           59       0.3677   10.86697
------------------------------------------------------------------------------
* lambda selected by cross-validation in final adaptive step.

. estimates store linearadaptive
```

Adaptive lasso performs multiple lassos. In the first lasso, a λ^* is selected, and penalty weights are constructed from the coefficient estimates. Then these weights are used in a second lasso, where another λ^* is selected. We did not specify how many lassos should be performed, so we got the default of two. We could specify more, but typically the selected λ^* does not change after the second lasso, or it changes little. See the `selection(adaptive)` option in [LASSO] **lasso**.

We can see details of the two lassos by using `lassoknots` and specifying the option `steps` to see all steps of the adaptive lasso.

```
. lassoknots, steps
```

Step	ID	lambda	No. of nonzero coef.	CV mean pred. error	Variables (A)dded, (R)emoved, or left (U)nchanged
1					
	1	.9109571	4	17.012	A 1.q3 1.q4 1.q5 1.gender
	2	.8300302	7	16.91096	A 0.q19 0.q85 3.q156
	3	.7562926	8	16.66328	A 0.q101
	4	.6891057	9	16.33224	A 0.q88
(*output omitted*)					
	23	.1176546	74	12.35715	A 3.q6 0.q40 0.q82 0.q98 0.q128 2.q134 0.q148 q157
	24	.1072025	78	12.40012	A 2.q6 0.q9 1.q34 4.q155
2					
	25	51.68486	4	17.01083	A 1.q3 1.q4 1.q5 1.gender
	26	47.09332	6	16.94087	A 0.q19 0.q85
(*output omitted*)					
	76	.4495285	45	10.33954	A 0.q44
	77	.4095937	46	10.33691	A q111
*	78	.3732065	46	10.33306	U
	79	.3400519	47	10.33653	A 0.q97
	80	.3098426	48	10.3438	A 0.q138
(*output omitted*)					
	112	.0157838	59	10.7663	A q70
	124	.0051685	59	10.86697	U

```
* lambda selected by cross-validation in final adaptive step.
```

Notice how the scale of λ changes in the second lasso. That is because of the penalty weights generated by the first lasso.

The ordinary lasso selected 49 variables, and the adaptive lasso selected 46. It is natural to ask how much these two groups of variables overlap. When the goal is prediction, however, we are not supposed to care about this. Ordinary lasso might select one variable, and adaptive lasso might instead select another that is highly correlated to it. So it is wrong to place importance on any particular variable selected or not selected. It is the group of variables selected as a whole that matters.

Still, we cannot resist looking, and the `lassocoef` command was designed especially for this purpose. We specify `lassocoef` with the option `sort(coef, standardized)`. This sorts the listing by the absolute values of the standardized coefficients with the largest displayed first. `lassocoef` can list different types of coefficients and display them in different orderings. See [LASSO] **lassocoef**.

```
. lassocoef linearcv linearadaptive, sort(coef, standardized)
```

	linearcv	linearadaptive
q19		
No	x	x
q85		
No	x	x
q5		
Yes	x	x
3.q156	x	x
q101		
No	x	x
(*output omitted*)		
q160		
No	x	x
age	x	x
q53	x	x
2.q105	x	
q102		
No	x	x
q154		
No	x	x
q111	x	x
q142		
No	x	x
0.q55	x	
0.q97	x	
q65		
4	x	x
1.q110	x	x
q70	x	
_cons	x	x
q44		
No		x

```
Legend:
  b - base level
  e - empty cell
  o - omitted
  x - estimated
```

We see that the adaptive lasso did not select four variables that the lasso did, and it selected one that the lasso did not. All the differences occurred among the variables with smaller standardized coefficients.

The most important question to ask is which performed better for out-of-sample prediction. `lassogof` is the command for that. We specify the `over()` option with the name of our sample indicator variable, `sample`. We specify the `postselection` option because for linear models, postselection

coefficients are theoretically slightly better for prediction than the penalized coefficients (which `lassogof` uses by default).

```
. lassogof linearcv linearadaptive, over(sample) postselection
Postselection coefficients
```

Name	sample	MSE	R-squared	Obs
linearcv				
	Training	8.652771	0.5065	503
	Testing	14.58354	0.2658	493
linearadaptive				
	Training	8.637575	0.5057	504
	Testing	14.70756	0.2595	494

The ordinary lasso did a little better in this case than the adaptive lasso.

Cross-validation folds

CV works by dividing the data randomly into K folds. One fold is chosen, and then a linear regression is fit on the other $K-1$ folds using the variables in the model for that λ. Then using these new coefficient estimates, a prediction is computed for the data of the chosen fold. The mean squared error (MSE) of the prediction is computed. This process is repeated for the other $K-1$ folds. The K MSEs are then averaged to give the value of the CV function.

Let's increase the number of folds from the default of 10 to 20 by specifying `selection(cv, folds(20))`.

```
. lasso linear q104 ($idemographics) $ifactors $vlcontinuous
> if sample == 1, selection(cv, folds(20)) rseed(9999)
20-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .9109571   no. of nonzero coef. =       4
Folds: 1...5....10....15....20   CVF = 17.08362
  (output omitted)
Grid value 23:    lambda = .1176546   no. of nonzero coef. =      74
Folds: 1...5....10....15....20   CVF = 12.12667
... cross-validation complete ... minimum found
Lasso linear model                          No. of obs        =        458
                                            No. of covariates =        273
Selection: Cross-validation                 No. of CV folds   =         20
```

	ID	Description	lambda	No. of nonzero coef.	Out-of-sample R-squared	CV mean prediction error
	1	first lambda	.9109571	4	0.0059	17.08362
	19	lambda before	.1706967	49	0.2999	12.03169
*	20	selected lambda	.1555325	55	0.3002	12.02673
	21	lambda after	.1417154	62	0.2988	12.05007
	23	last lambda	.1176546	74	0.2944	12.12667

```
* lambda selected by cross-validation.
. estimates store linearcv2
```

Which performs better for out-of-sample prediction?

```
. lassogof linearcv linearcv2, over(sample) postselection
Postselection coefficients
```

Name	sample	MSE	R-squared	Obs
linearcv				
	Training	8.652771	0.5065	503
	Testing	14.58354	0.2658	493
linearcv2				
	Training	8.545785	0.5126	502
	Testing	14.7507	0.2594	488

The first lasso with 10 folds did better than the lasso with 20 folds. This is generally true. More than 10 folds typically does not yield better predictions.

We should mention again that CV is a randomized procedure. Changing the random-number seed can result in a different λ^* being selected and so give different predictions.

BIC

We are now going to select λ^* by minimizing the BIC function, which we do by specifying the option `selection(bic)`.

```
. lasso linear q104 ($idemographics) $ifactors $vlcontinuous
> if sample == 1, selection(bic)
Evaluating up to 100 lambdas in grid ...
Grid value 1:     lambda = .9109571   no. of nonzero coef. =       4
                  BIC = 2618.642
Grid value 2:     lambda = .8300302   no. of nonzero coef. =       7
                  BIC = 2630.961
Grid value 3:     lambda = .7562926   no. of nonzero coef. =       8
                  BIC = 2626.254
Grid value 4:     lambda = .6891057   no. of nonzero coef. =       9
                  BIC = 2619.727
Grid value 5:     lambda = .6278874   no. of nonzero coef. =      10
                  BIC = 2611.577
Grid value 6:     lambda = .5721076   no. of nonzero coef. =      13
                  BIC = 2614.155
Grid value 7:     lambda = .5212832   no. of nonzero coef. =      13
                  BIC = 2597.164
Grid value 8:     lambda = .4749738   no. of nonzero coef. =      14
                  BIC = 2588.189
Grid value 9:     lambda = .4327784   no. of nonzero coef. =      16
                  BIC = 2584.638
Grid value 10:    lambda = .3943316   no. of nonzero coef. =      18
                  BIC = 2580.891
Grid value 11:    lambda = .3593003   no. of nonzero coef. =      22
                  BIC = 2588.984
Grid value 12:    lambda =  .327381   no. of nonzero coef. =      26
                  BIC = 2596.792
Grid value 13:    lambda = .2982974   no. of nonzero coef. =      27
                  BIC = 2586.521
Grid value 14:    lambda = .2717975   no. of nonzero coef. =      28
                  BIC = 2578.211
Grid value 15:    lambda = .2476517   no. of nonzero coef. =      32
                  BIC = 2589.632
```

```
Grid value 16:    lambda =  .225651   no. of nonzero coef. =      35
                  BIC = 2593.753
Grid value 17:    lambda = .2056048   no. of nonzero coef. =      37
                  BIC = 2592.923
Grid value 18:    lambda = .1873395   no. of nonzero coef. =      42
                  BIC = 2609.975
Grid value 19:    lambda = .1706967   no. of nonzero coef. =      49
                  BIC = 2639.437
... selection BIC complete ... minimum found
Lasso linear model                          No. of obs        =        458
                                            No. of covariates =        273
Selection: Bayesian information criterion
```

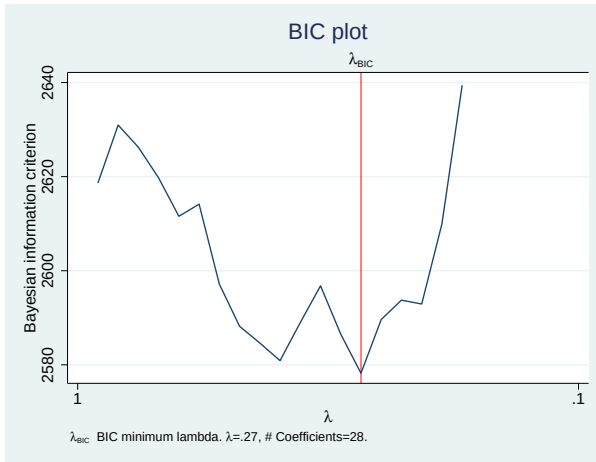
ID	Description	lambda	No. of nonzero coef.	In-sample R-squared	BIC
1	first lambda	.9109571	4	0.0308	2618.642
13	lambda before	.2982974	27	0.3357	2586.521
* 14	selected lambda	.2717975	28	0.3563	2578.211
15	lambda after	.2476517	32	0.3745	2589.632
19	last lambda	.1706967	49	0.4445	2639.437

```
* lambda selected by Bayesian information criterion
. estimates store linearbic
```

The minimum of the BIC function was found to be at $\lambda = 0.272$. It selects λ^* as this λ, which corresponds to 28 variables in the model out of 273 potential variables.

After fitting a lasso using BIC, it is a good idea to plot the BIC function and look at the shape of the curve around the minimum.

```
. bicplot
```



We see that the BIC function rises sharply once it hits the minimum. By default, the `lasso` command stops when it has identified a minimum.

So far, we have fit lasso linear models using CV, an adaptive lasso, and BIC. Which one performs better in the out-of-sample prediction?

```
. lassogof linearcv linearadaptive linearbic, over(sample) postselection
Postselection coefficients
```

Name	sample	MSE	R-squared	Obs
linearcv				
	Training	8.652771	0.5065	503
	Testing	14.58354	0.2658	493
linearadaptive				
	Training	8.637575	0.5057	504
	Testing	14.70756	0.2595	494
linearbic				
	Training	9.740229	0.4421	508
	Testing	13.44496	0.3168	503

The BIC lasso performs the best.

More potential variables than observations

Lasso has no difficulty fitting models when the number of potential variables exceeds the number of observations.

We use `vl substitute` to create interactions of all of our factor-variable indicators with our continuous variables.

```
. vl substitute interact = i.factors##c.vlcontinuous
```

We fit the lasso.

```
. lasso linear q104 ($idemographics) $interact if sample == 1, rseed(1234)
note: 1.q32#c.q70 omitted because of collinearity with another variable.
note: 2.q34#c.q63 omitted because of collinearity with another variable.
  (output omitted)
10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = 1.020853   no. of nonzero coef. =       4
Folds: 1...5....10   CVF = 16.93478
  (output omitted)
Grid value 34:    lambda = .2199361   no. of nonzero coef. =     106
Folds: 1...5....10   CVF = 12.91285
... cross-validation complete ... minimum found
Lasso linear model                          No. of obs        =        458
                                            No. of covariates =      7,223
Selection: Cross-validation                 No. of CV folds   =         10
```

ID	Description	lambda	No. of nonzero coef.	Out-of-sample R-squared	CV mean prediction error
1	first lambda	1.020853	4	0.0146	16.93478
29	lambda before	.2775279	80	0.2531	12.83525
* 30	selected lambda	.2649138	85	0.2545	12.81191
31	lambda after	.2528731	89	0.2541	12.81893
34	last lambda	.2199361	106	0.2486	12.91285

```
* lambda selected by cross-validation.
. estimates store big
```

There were 7,223 potential covariates in our model, of which lasso selected 85. That seems significantly more than the 49 selected by our earlier lasso.

Let's see how they do for out-of-sample prediction.

```
. lassogof linearcv big, over(sample) postselection
Postselection coefficients
```

Name	sample	MSE	R-squared	Obs
linearcv				
	Training	8.652771	0.5065	503
	Testing	14.58354	0.2658	493
big				
	Training	6.705183	0.6117	490
	Testing	17.00972	0.1403	478

Our model with thousands of potential covariates did better for in-sample prediction but significantly worse for out-of-sample prediction.

Factor variables in lasso

It is important to understand how lasso handles factor variables. Let's say we have a variable, `region`, that has four categories representing four different regions of the country. Other Stata estimation commands handle factor variables by setting one of the categories to be the base level; it then makes indicator variables for the other three categories, and they become covariates for the estimation.

Lasso does not set a base level. It creates indicator variables for all levels (`1.region`, `2.region`, `3.region`, and `4.region`) and adds these to the set of potential covariates. The reason for this should be clear. What if `1.region` versus the other three categories is all that matters for prediction? Lasso would select `1.region` and not select the other three indicators. If, however, `1.region` was set as a base level and omitted from the set of potential covariates, then lasso would have to select `2.region`, `3.region`, and `4.region` to pick up the `1.region` effect. It might be wasting extra penalty on three coefficients when only one was needed.

See [LASSO] **Collinear covariates**.

Lasso logit and probit models

lasso will also fit logit, probit, and Poisson models.

We fit a logit model.

```
. lasso logit q106 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234)

10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .1155342   no. of nonzero coef. =       0
Folds: 1...5....10   CVF = 1.384878
  (output omitted)
Grid value 27:    lambda =  .010285   no. of nonzero coef. =      88
Folds: 1...5....10   CVF = 1.147343
... cross-validation complete ... minimum found

Lasso logit model                         No. of obs        =        458
                                          No. of covariates =        277
Selection: Cross-validation               No. of CV folds   =         10
```

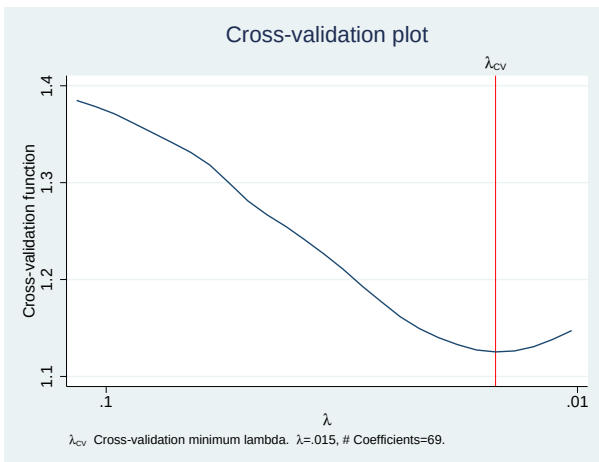
ID	Description	lambda	No. of nonzero coef.	Out-of-sample dev. ratio	CV mean deviance
1	first lambda	.1155342	0	-0.0004	1.384878
22	lambda before	.0163767	65	0.1857	1.127315
* 23	selected lambda	.0149218	69	0.1871	1.125331
24	lambda after	.0135962	73	0.1864	1.126333
27	last lambda	.010285	88	0.1712	1.147343

```
* lambda selected by cross-validation.

. estimates store logit
```

Logit and probit lasso models are famous for having CV functions that are more wiggly than those for linear models.

```
. cvplot
```



This curve is not as smoothly convex as was the CV function for the linear lasso shown earlier. But it is not as bad as some logit CV functions. Because the CV functions for nonlinear models are not as smooth, `lasso` has a stricter criterion for declaring that a minimum of the CV function is found than it has for linear models. `lasso` requires that five smaller λ's to the right of a nominal minimum be observed with larger CV function values by a relative difference of `cvtolerance(#)` or more. Linear models only require three such λ's be found before declaring a minimum and stopping.

Let's now fit a probit model.

```
. lasso probit q106 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234)

10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .1844415   no. of nonzero coef. =       0
Folds: 1...5....10   CVF = 1.384877
  (output omitted)
Grid value 26:    lambda = .0180201   no. of nonzero coef. =      87
Folds: 1...5....10   CVF = 1.152188
... cross-validation complete ... minimum found

Lasso probit model                          No. of obs        =        458
                                            No. of covariates =        277
Selection: Cross-validation                 No. of CV folds   =         10
```

ID	Description	lambda	No. of nonzero coef.	Out-of-sample dev. ratio	CV mean deviance
1	first lambda	.1844415	0	-0.0004	1.384877
21	lambda before	.0286931	61	0.1820	1.132461
* 22	selected lambda	.0261441	64	0.1846	1.128895
23	lambda after	.0238215	70	0.1841	1.129499
26	last lambda	.0180201	87	0.1677	1.152188

```
* lambda selected by cross-validation.

. estimates store probit
```

`lassocoef` can be used to display coefficient values. Obviously, logit and probit coefficient values cannot be compared directly. But we do see similar relative scales.

```
. lassocoef logit probit, sort(coef, standardized) display(coef, standardized)
```

	logit	probit
q142		
No	-.50418	-.3065817
q154		
No	-.3875702	-.2344515
q90		
No	-.3771052	-.2288992
q8		
No	-.3263827	-.200673
(output omitted)		
q37		
No	-.0128537	-.0062874
2.q158	.0065661	.0012856
3.q65	-.0062113	
3.q110	-.0055616	
q120	.0044864	
0.q146	-.004312	
q95		
3	.0030261	

```
Legend:
  b - base level
  e - empty cell
  o - omitted
```

The probit lasso selected five fewer variables than logit, and they were the five variables with the smallest absolute values of standardized coefficients.

We look at how they did for out-of-sample prediction.

```
. lassogof logit probit, over(sample)
Penalized coefficients
```

Name	sample	Deviance	Deviance ratio	Obs
logit				
	Training	.8768969	0.3674	499
	Testing	1.268346	0.0844	502
probit				
	Training	.8833892	0.3627	500
	Testing	1.27267	0.0812	503

Neither did very well. The out-of-sample deviance ratios were notably worse than the in-sample values. The deviance ratio for nonlinear models is analogous to R^2 for linear models. See *Methods and formulas* for [LASSO] **lassogof** for the formal definition.

We did not specify the postselection option in this case because there are no theoretical grounds for using postselection coefficients for prediction with nonlinear models.

Lasso Poisson models

Finally, we fit a Poisson model.

```
. lasso poisson q107 $idemographics $ifactors $vlcontinuous
> if sample == 1, rseed(1234)

10-fold cross-validation with 100 lambdas ...
Grid value 1:     lambda = .5745539   no. of nonzero coef. =       0
Folds: 1...5....10   CVF = 2.049149
  (output omitted)
Grid value 21:    lambda =  .089382   no. of nonzero coef. =      66
Folds: 1...5....10   CVF = 1.653376
... cross-validation complete ... minimum found

Lasso Poisson model                          No. of obs        =        458
                                             No. of covariates =        277
Selection: Cross-validation                  No. of CV folds   =         10
```

ID	Description	lambda	No. of nonzero coef.	Out-of-sample dev. ratio	CV mean deviance
1	first lambda	.5745539	0	-0.0069	2.049149
16	lambda before	.1423214	37	0.1995	1.629222
* 17	selected lambda	.129678	45	0.1999	1.628315
18	lambda after	.1181577	48	0.1993	1.62962
21	last lambda	.089382	66	0.1876	1.653376

```
* lambda selected by cross-validation.
```

We see how it does for out-of-sample prediction.

```
. lassogof, over(sample)
Penalized coefficients
```

sample	Deviance	Deviance ratio	Obs
Training	1.289175	0.3515	510
Testing	1.547816	0.2480	502

Its in-sample and out-of-sample predictions are fairly close. Much closer than they were for the logit and probit models.

Also see

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

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