Computational Statistics 12.2: Regression with shrinkage

Lasso and Ridge Regression

Shrinkage
- **Lasso**
- **Ridge regression and elastic net**

Example
- **Example, ridge**
- **Example, lasso**
- **Choosing lambda**
- **Comparison with regression**

<

>

Lasso and Ridge Regression

Overview

This lesson introduces the lasso and ridge regression shrinkage methods.

Objectives

After completing this module, students should be able to:

1. Explain how shrinkage methods work to reduce the number of variables in a model.
2. Distinguish between lasso, ridge regression, and elastic net.
3. Apply the glmnet package to estimate models with many independent variables.
4. Test and validate those models out-of-sample.

Readings

Lander, 19.1

Recommended: James et al, An Introduction to Statistical Learning (http://www-bcf.usc.edu/~gareth/ISL/ISLR%20Fourth%20Printing.pdf), Chapter 6.2.

Shrinkage

In the previous module we have already seen one potential method for reducing a large number of X variables to few variables: we could do a factor analysis on all those variables, and replace the X variables with the fewer factors. Again, each new variable Z would be the projection of that factor on the observations: that is, the score for each observation on that factor (eg, observation i might be a 1.3 on the factor that ranges between sleepy and energetic).

However, this approach, although perfectly good and very popular, does have the drawback of replacing our existing and well-understood variables with the factors, which might be a bit harder to interpret and might not capture as much of the variation in the data as the original variables did. A different approach might be to better select which subset of the variables actually influence y, and perhaps even weight them according to which ones have a greater or lesser contribution to explaining and predicting y. In a sense, this generalizes what we have always done in multiple regression: in multiple regression, some of the β coefficients are significant, and we might include those in the final regression if we were doing some sort of stepwise variable selection; whereas other β coefficients are not statistically significant, and we declare them to be equivalent to 0.

With *shrinkage* methods, we take all of perhaps hundreds of β coefficients and declare some 0, some their original estimated value, and others – that are significant, say, but not very significant – something inbetween their estimated value and 0. That is, instead of having such a hard cut-off – which always seems a bit odd given the weirdly hard line at p=0.05 – we have a more gradual line, where the most significant β coefficients stay as they would be from the regression, and others are shrunk more or less towards 0 depending on their significance levels. For all of you who were worrying about the artibrariness of the p=0.05 cutoff, this approach might make you somewhat happier, and is closer to the "bayesian" mindset we discussed earlier.

A second advantage of shrinkage methods – and perhaps a more valuable one than merely theoretical – is that it actually allows us to use quite a few more independent variables without necessarily over-fitting y. Since the shrinkage rate – how severely β coefficients with high p values (ie, the least significant ones) – is something we can set ourselves, we can simply choose the best shrinkage rate given our data to maximize our out-of-sample accuracy. If keeping too many coefficients unshrunk gives too much over-fitting and bad out-of-sample accuracy, we can can just increase the shrinkage rate until only a few coefficients – the most statistically significant ones – are retained, giving a smaller model that more accuratly extrapolates to the out-sample test.

Lasso

There are two similar methods for shrinkage, *lasso* and *ridge regression*, that have different historical origins but very similar structures.

As you may recall, the basic idea in regular multiple regression is to choose the β coefficients that maximize the match between $\hat{y}$ and y. That is, we want to choose the β coefficients to minimize the following sum:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1 - \beta_2 x_2 \dots)^2$$

The solution to this minimization problem is what gives us our formula for calculating our β coefficients from our dataset of y and x variables.

The lasso shrinkage model is very similar to this, with one addition. Instead of wanting to minimize the above sum, we instead want to minimize:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1 - \beta_2 x_2 \dots)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

That is, instead of just optimizing the fit between $\hat{y}$ and y (which makes the sum in the first equation as small as possible), we now want to minimize that sum while also minimizing the sum of the absolute values of all the β coefficients (except the constant). This introduces a trade-off: on the one hand, if we keep all the β coefficients at the values as estimated by the simple regression (top equation), we optimize the fit between $\hat{y}$ and y, but we also have a large $\sum_{j=1}^{p} |\beta_j|$ penalty; on the other hand, we can reduce that second penalty to 0 by shrinking all the β coefficients to 0, but at the cost of worsening the fit between $\hat{y}$ and y (and thus increasing the first sum). The value λ sets the tradeoff between the two terms: when λ is 0, we have our usual multiple regression equation with entirely un-shrunk coefficients; and when λ is very large, then the first sum plays almost no role and the equation is governed by the second sum, leading to most or all β coefficients being shrunk to 0.

You might ask, if the multiple regression coefficients are the ones that produce the best fit between $\hat{y}$ and y, why wouldn't we want to use those? But the answer lies in over-fitting: sometimes it is worth getting a worse fit between $\hat{y}$ and y in-sample, because it produces a better fit out-of-sample – and thus is better capturing the true population model, rather than just some random noise. But to get this better out-of-sample validity, we often have to try a bunch of λ values to find the one that works best out-of-sample.

In addition, the shrinkage reduced the number of non-0 variables, which means that both statistically insignificant ones and (depending on λ) the less significant ones get shrunk to 0, leaving many fewer to interpret.

Ridge regression and elastic net

The second variant is ridge regression, which looks very similar to lasso:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1 - \beta_2 x_2 \dots)^2 + \lambda \sum_{j=1}^{p} (\beta_j)^2$$

This too penalizes the model for large β coefficients, shrinking them down towards 0. The practical difference is that, while lasso tends to shrink many β coefficients all the way to 0, ridge regression shrinks most of them towards 0, but few all the way to 0. This means that the lasso results are sometimes easier to interpret, because you have overall fewer non-zero coefficients.

One last option is to combine the two into a single shrinkage model, called *elastic net*:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1 - \beta_2 x_2 \dots)^2 + \lambda(\alpha \sum_{j=1}^{p} |\beta_j| + (1-\alpha)\sum_{j=1}^{p}(\beta_j)^2)$$

As you can see, this once again has a single parameter λ which sets the total amount of shrinkage, plus another parameter α that sets the proportion between the lasso and ridge amount (α ranges between 0 and 1, so the proportion can be anywhere between 100% lasso to 50/50 to 100% ridge).

Example

We use here an example from *An Introduction to Statistical Learning with Applications in R* (ISLR), which is an excellent book (and can be found online here (http://www-bcf.usc.edu/~gareth/ISL/)) and happens to have been written by some of the creators of elastic net and the R package we will be using, `glmnet`.

We use a dataset from ISLR, on various baseball players, including their performance levels and salaries:

```
#install.packages("glmnet")
#install.packages("ISLR") #datasets
library(ISLR)
data(Hitters)
dim(Hitters)
```

```
[1] 322  20
```

```
head(Hitters)
```

```
                 AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits
-Andy Allanson     293   66     1   30  29    14     1    293    66
-Alan Ashby        315   81     7   24  38    39    14   3449   835
-Alvin Davis       479  130    18   66  72    76     3   1624   457
-Andre Dawson      496  141    20   65  78    37    11   5628  1575
-Andres Galarraga  321   87    10   39  42    30     2    396   101
-Alfredo Griffin   594  169     4   74  51    35    11   4408  1133
                 CHmRun CRuns CRBI CWalks League Division PutOuts Assists
-Andy Allanson        1    30   29     14      A        E     446      33
-Alan Ashby          69   321  414    375      N        W     632      43
-Alvin Davis         63   224  266    263      A        W     880      82
-Andre Dawson       225   828  838    354      N        E     200      11
-Andres Galarraga    12    48   46     33      N        E     805      40
-Alfredo Griffin     19   501  336    194      A        W     282     421
                 Errors Salary NewLeague
-Andy Allanson       20     NA         A
-Alan Ashby          10  475.0         N
-Alvin Davis         14  480.0         A
-Andre Dawson          3  500.0         N
-Andres Galarraga     4   91.5         N
-Alfredo Griffin     25  750.0         A
```

As we can see, there are 20 variables in this dataset and 322 observations. We can use elastic net to model salaries as a function of the players' various skills and achievements. `glmnet` works much like regression, but unlike `lm` it doesn't understand factors, so it expects only a matrix of numeric variables. If you have factors, they can first be converted to dummies, using for instance `model.matrix` or `build.x` in the `useful` package (see Lander or the ISLR book for more on this). But for our purposes here, we'll just extract the variables from Hitters that are numeric (not including the dependent variable, Salary), and use that:

```
Hitters <- na.omit(Hitters)
numericvars <- c(1:13,16:18)
x <- as.matrix(Hitters[,numericvars])
y <- Hitters$Salary
```

`glmnet` lets you set your alpha level for the proportion of lasso vs ridge regression; alpha=0 means pure ridge, and alpha=1 means pure lasso. Since the key question is setting the appropriate shrinkage level λ (lambda), glmnet is designed to run repeatedly on a large number of different lambda settings, after which you can predict $\hat{y}$ for each shrinkage level, and see which predicted $\hat{y}$ best matches the real y values out-of-sample.

So next we set the levels of lambda (shrinkage) to tell glmnet to try, ranging from a very high lambda to a very low lambda:

```
lambdalevels <- 10^seq(7,-2,length=100)
```

This gives us a 100-item sequence ranging from 10^7 down to 10^{-2}, which will cause glmnet to run the model 100 times, once with each of these lambda values.

Example, ridge

First let's try it with alpha=0, ie, pure ridge. The formula is simple enough:

```
library(glmnet)
ridge.mod=glmnet(x,y,alpha=0,lambda=lambdalevels)
```

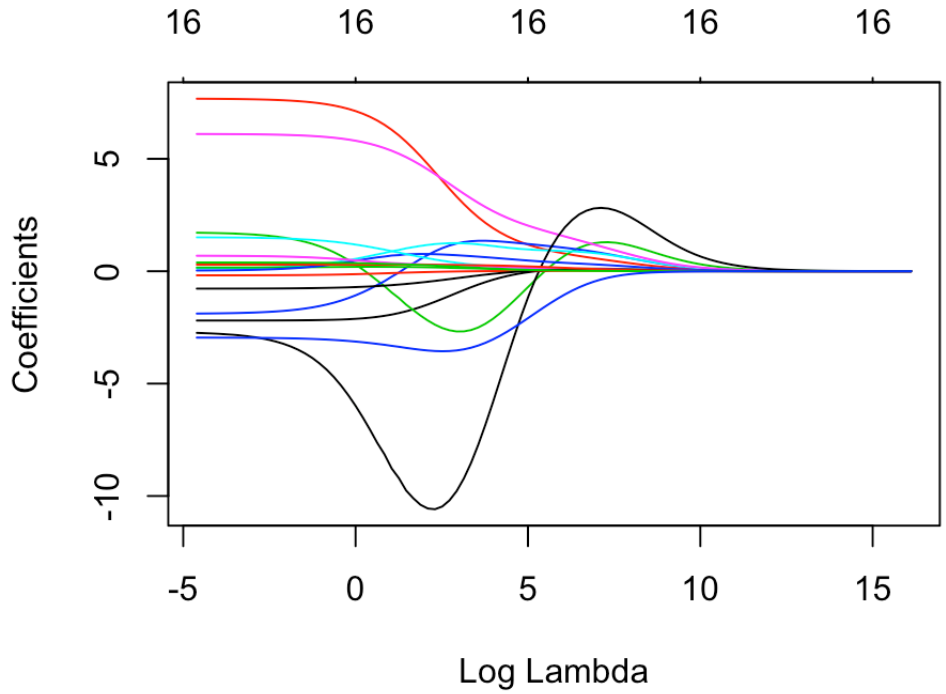
glmnet stores the β coefficients for every lambda level, so there are p x #-of-lambda-levels coeffcients stored. To see the 100th set of coefficients (for the smallest lambda), we can write:

```
coef(ridge.mod)[,100]
```

```
 (Intercept)        AtBat         Hits        HmRun         Runs
126.74245930  -2.19339700   7.67471127   1.71439290  -1.88406702
         RBI        Walks        Years       CAtBat        CHits
  0.12400003   6.10591224  -2.74289815  -0.18258474   0.16444736
      CHmRun        CRuns         CRBI       CWalks      PutOuts
  0.03237573   1.50756927   0.68881161  -0.77369053   0.29461825
     Assists       Errors
  0.38724591  -2.95337311
```

We can also see how those various β coefficients were shrunk down as we increase lambda from a small level of shrinkage (lambda=10^{-2}) to a large level (lambda=10^7).

```
plot(ridge.mod,xvar="lambda")
```

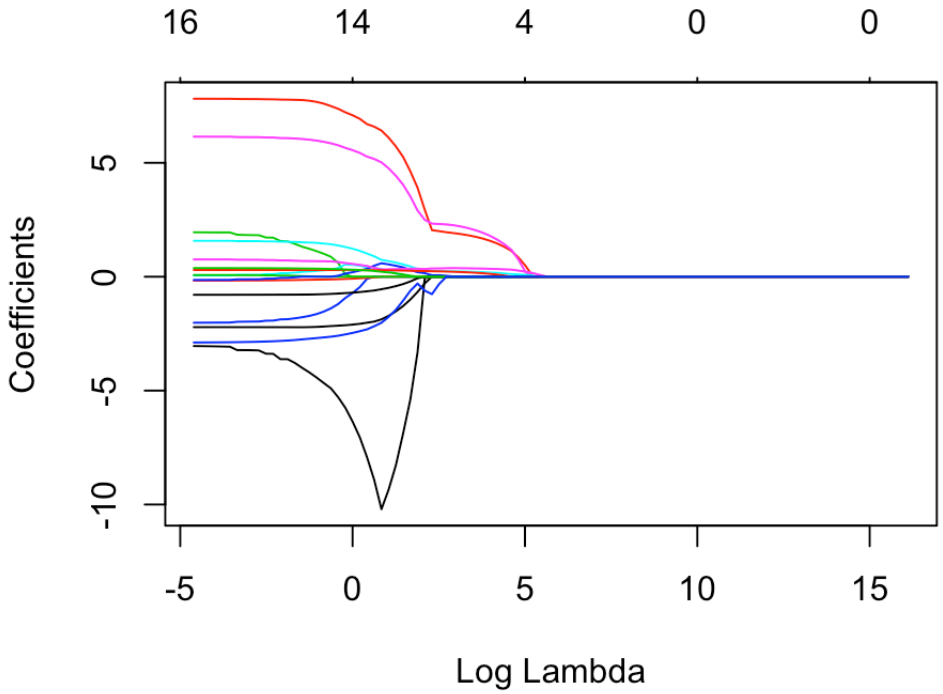


The coefficient values are plotted along the y axis, and the (natural log) lambda value along the x; a vertical slice for instance at log lambda=0 gives the values of the 16 coefficientsat that level of shrinkage. One thing you can discern is that "shrinkage" is too simple a term: some coefficients actually can briefly grow in value for some levels of λ if that minimizes the sum in the equation we defined above, although all will eventually shrink to 0 when lambda is large enough.

Example, lasso

We can do the same for a pure lasso model:

```
lasso.mod=glmnet(x,y,alpha=1,lambda=lambdalevels)
plot(lasso.mod,xvar="lambda")
```



As can be seen, the results are not too dissimilar for these data.

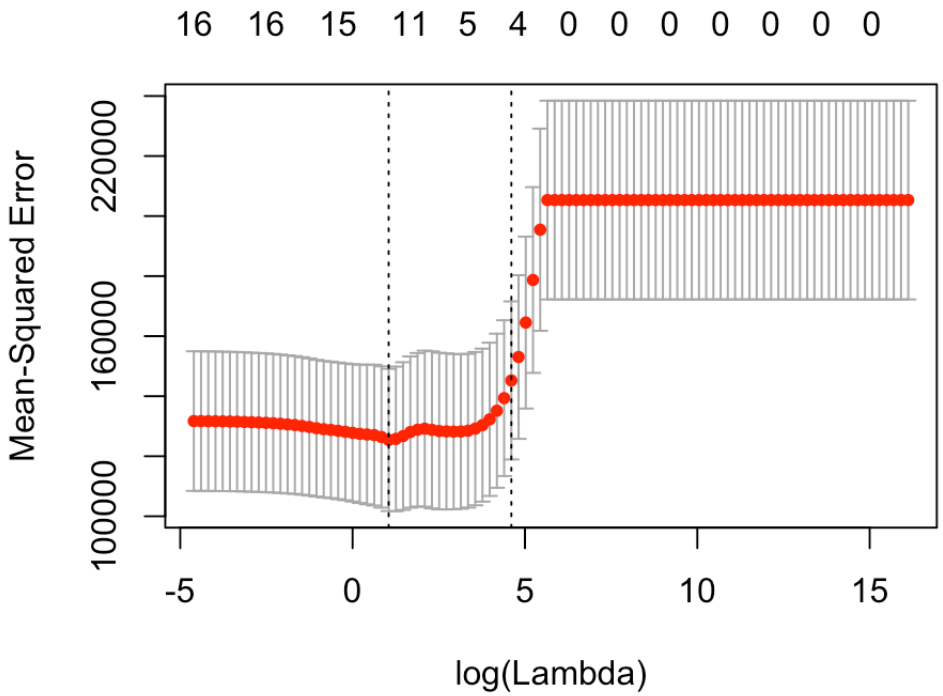
Choosing lambda

Of course, what we most want is the value of lambda that produces the best out-of-sample fit – ie, that minimizes the out-of-sample mean squared error (MSE) between $\hat{y}$ and y. `glmnet` therefore has built into it a cross-validation procedure, where it does k-fold cross validation for every level of lambda and calculated the MSE for each, and chooses the lambda with the lowest error. That is:

1. The data are divided into k portions.
2. The model is fit on the in-sample (all but portion k), and for each level of lambda, $\hat{y}$ is predicted on the out-sample and the MSE for that lambda level is calculated.
3. Thus for each level of lambda there are k MSE values, one for each of the k out-of-sample tests.

We can then plot the average MSE for each lambda level (red dot), along with the 95% CI for that MSE (because we have k of them for each lambda level):

```
set.seed(1)
cv.lasso.mod=cv.glmnet(x,y,alpha=1,lambda=lambdalevels)
plot(cv.lasso.mod)
```



The dotted vertical line on the left shows the lambda that gets the best MSE, and the dotted line on the right is the MSE that is 1 standard error larger. The values along the top are the number of non-zero coefficients (after shrinkage). As we can see for this data, the best fit is with about 12 coefficients, but we do only one standard error worse (ie, statistically indistinguishable) with a model with only about 4 non-zero coefficients. Thus we can choose either the model with the best out-of-sample accuracy with 12 coefficients, or an only slightly-worse but much simpler model with 4 coefficients.

If we want to know the coefficients for the best lambda, we can get the best lambda and plug it into `predict` with `s=` for the best lambda and `type="coefficients"` to get the coefficients:

```
bestlambda <- cv.lasso.mod$lambda.min
predict(lasso.mod, type="coefficients",s=bestlambda)
```

```
17 x 1 sparse Matrix of class "dgCMatrix"
                     1
(Intercept) 76.4896752
AtBat       -1.7145436
Hits         6.1037612
HmRun        .
Runs         .
RBI          0.2484785
Walks        4.7562058
Years       -9.3407580
CAtBat       .
CHits        .
CHmRun       0.5407148
CRuns        0.6858672
CRBI         0.3235562
CWalks      -0.4882173
PutOuts      0.2800842
Assists      0.1852786
Errors      -1.7645945
```

Note how four have been shrunk to 0.

Comparison with regression

So finally, let's see how the shrinkage model compares to regular multiple regression. Imagine we are in a contest: we are given one subset of the data, the training set. We can do whatever we want with it to fit the model, and then we are given a second set of data, the test set. We use our model from the training set to predict $\hat{y}$ in the test set, and whichever model has the best MSE between its $\hat{y}$ and the true y wins (and may in fact be the more "true" model).

First we divide our data randomly into two equal halves, the training set and the test set:

```
set.seed(1)
train <- sample (1:nrow(x),nrow(x)/2)
test <- (-train)
trainx <- x[train,]
testx <- x[test,]
trainy <- y[train]
testy <- y[test]
```

First let's try regession. We fit the model on the training set, use the test data to predict $\hat{y}$, and then calculate the MSE of that vs the true test y:

```
lmout <- lm(trainy~trainx)
yhat.r <- cbind(1,testx) %*% lmout$coefficients
# ^^ we predict via matrix multiplication rather than just using predict()
#    because x was made a matrix to make glmnet happy
mse.reg <- sum((testy - yhat.r)^2)/nrow(testx)
mse.reg
```

```
[1] 128137
```

Now let's do it with lasso:

```
cv.lasso.mod=cv.glmnet(trainx,trainy,alpha=1,lambda=lambdalevels)
yhat.l <- predict(cv.lasso.mod$glmnet.fit, s=cv.lasso.mod$lambda.min, newx=testx)
# ^^ note how we give predict() our best lambda.min value to use for prediction
mse.las <- sum((testy - yhat.l)^2)/nrow(testx)
mse.las
```

```
[1] 106427.9
```

So the lasso MSE is better. Is that difference a lot? Well, we can construct an number like R^2 to compare: we construct the total sum of squares (TSS) as the error between `testy` and the mean of `trainy` – ie, as our dumbest model, we just guess the mean of our training set y values as our $\hat{y}$ guess for every new y in the test set. We then compare the sum of squared errors `sum((testy - yhat)^2)` for our other two methods (regression and lasso), and construct an R^2-like measure for each. (It's not exactly R^2 even for the regression because of course we are comparing $\hat{y}$ as predicted from one sample with y in a different sample, rather than all in the same sample as we do with the proper R^2).

```
tss <- sum((testy - mean(trainy))^2)
sse.reg <- sum((testy - yhat.r)^2)
sse.las <- sum((testy - yhat.l)^2)
r2.r <- (tss - sse.reg) / tss
r2.l <- (tss - sse.las) / tss
r2.r
```

```
[1] 0.3369474
```

```
r2.l
```

```
[1] 0.4492823
```

As we can see, the out-of-sample pseudo-R^2 is considerably higher for the lasso than the regression method. Many other tests have established lasso and ridge regression (and elastic net more generally) as some of the most successful out-of-sample modeling methods. It is likely that the lasso model is doing a better job of fitting the real population, and not over-fitting due to having a high number of independent variables, the way the regular multiple regression is prone to do. Similarly, we might reasonably believe that the lasso coefficients in the previous slide are likely to be more accurate measures of the true effects than the coefficients we get from just running the `lm` regression.