Lab 4: One last time through the hot New Jersey night

Ben Thomas

10/7/2019

Ridge and lasso regressions

Previously, in Lab 3, Olek Wojcik and I used regular linear regression to predict the number of violent crimes per person. I'll now try to do this using ridge and lasso regressions, to see if adding constraints can decrease the overall MSE.

Lasso model

First, let's download the data and assign x and y values:

```
d <- read.csv("http://andrewpbray.github.io/data/crime-train.csv")
x <- model.matrix(ViolentCrimesPerPop~., d)[,-1]
y <- d$ViolentCrimesPerPop
```

Then, let's use the glmnet() package to fit a lasso regression model, with a 100-long sequence of lambdas.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```
# Construct the model
grid=10^seq(10,-2,length=100)
lasso.mod=glmnet(x,y,alpha=1,lambda=grid)

# Take a look at the dimensions of our coefficients
dim(coef(lasso.mod))
```

```
## [1] 2144  100
```

Note that the number of coefficients in this model seems huge, maybe reflecting the weird setup of "communities" in the original dataset. I'll return to this point later.

```
# Find the optimal lambda
set.seed(1)
cv.out = cv.glmnet(x, y, alpha=1)
lasso.lam = cv.out$lambda.min

# Determine the number of coefficients in the lasso model with our optimal lambda value
out = glmnet(x, y, alpha=1, lambda=grid)
lasso.coef = predict(out, type = "coefficients", s = lasso.lam)[1:2144,]
nonzero.lasso.coef <- lasso.coef[lasso.coef != 0]
length(nonzero.lasso.coef)
```

```
## [1] 87
```

As we can see here, the number of coefficients in our optimal model is apparently 87 (including the intercept). Before moving on, let's take a quick look at what these coefficients are:

```
nonzero.lasso.coef[1:15]
```

```
##                    (Intercept)                          state
##                   0.3392755995                 -0.0007124188
##                       county35                       county99
##                  -0.0011476773                  0.0457169431
##                  community13045                 community21990
##                   0.0032937885                  0.1487883290
##                  community23875                  community2480
##                   0.0242788480                  0.0008649811
##                  community34550                 community43554
##                   0.0001040461                  0.1403429196
##                  community88000     communitynameAndersoncity
##                   0.0000318232                  0.0068640904
##  communitynameArkadelphiacity communitynameBalchSpringscity
##                  -0.0060875716                  0.1808571854
##  communitynameBlythevillecity
##                   0.0247720177
```

All of the first 15 coefficients have something to do with the location of the town in question; not exactly what I'd initially think is important. This, again, might have something to do with the fact that each of the different communities is coded as a different variable, somewhat exploding the amount of predictors.

```
# Calculate the training MSE
lasso.pred = predict(lasso.mod,s=lasso.lam, newx = x)
lasso.MSE <- mean((lasso.pred-y)^2)
lasso.MSE
```

```
## [1] 0.0153115
```

As seen above, the training MSE for the lasso model is 0.0153115.

Ridge model

Let's now run the ridge regression and see how its MSE compares.

```
grid=10^seq(10,-2,length=100)
ridge.mod=glmnet(x,y,alpha=0,lambda=grid)

# Find the best lambda value
set.seed(1)
ridge.cv.out=cv.glmnet(x, y, alpha=0)
ridge.lam=ridge.cv.out$lambda.min

# Find our training MSE for the ridge model using the best lambda value
ridge.pred=predict(ridge.mod,s=ridge.lam, newx = x)
ridge.MSE <- mean((ridge.pred-y)^2)
ridge.MSE
```

```
## [1] 0.01548108
```

The training MSE for the ridge regression, using our original data, is 0.01548108. This is very comparable to the one produced by the lasso model.

An aside

Using our original dataset, we had 2144 different coefficients in each of our models; probably far too many for us to interpret easily. Many of the coefficients result from the fact that there are many different communities. Additionally, many of the "Lemas" variables have missing data. As an interesting exercise (and to see if our results change) I'll trim down our original dataset to work around these two problems and try running the ridge and lasso regressions once again.

```
library(tidyverse)
# Trim the dataset
d_new <- d %>%
  select(population:PctSameState85,
         ViolentCrimesPerPop)

# Set up the new x and y
x_new <- model.matrix(ViolentCrimesPerPop~., d_new)[,-1]
y_new <- d_new$ViolentCrimesPerPop
```

```
# Run our new ridge regression
grid=10^seq(10,-2,length=100)
new.ridge.mod = glmnet(x_new, y_new, alpha=0, lambda=grid)
dim(coef(new.ridge.mod))
```

```
## [1]  97 100
```

This gives a much more reasonable 97 coefficients in the model. Let's now find the optimal lambda value using cross validation, and then calculate the training MSE of this new model.

```
# Find the best lambda value
set.seed(1)
new.ridge.cv.out=cv.glmnet(x_new, y_new, alpha=0)
new.ridge.lam=new.ridge.cv.out$lambda.min

# Find our training MSE for the ridge model using the best lambda value
new.ridge.pred=predict(new.ridge.mod,s=new.ridge.lam, newx = x_new)
new.ridge.MSE <- mean((new.ridge.pred-y_new)^2)
new.ridge.MSE
```

```
## [1] 0.01770891
```

The training MSE of this new ridge model is 0.01771; noticeably worse than the original ridge model's.

Now, let's construct the new lasso model.

```
# Construct the model
grid=10^seq(10,-2,length=100)
new.lasso.mod=glmnet(x_new,y_new,alpha=1,lambda=grid)

# Find the optimal lambda
set.seed(1)
new.lasso.cv.out=cv.glmnet(x_new, y_new, alpha=1)
new.lasso.lam=new.lasso.cv.out$lambda.min

# Calculate the training MSE
new.lasso.pred=predict(new.lasso.mod,s=new.lasso.lam, newx = x_new)
new.lasso.MSE <- mean((new.lasso.pred-y_new)^2)
new.lasso.MSE
```

```
## [1] 0.0186954
```

This model's training MSE is 0.01869, which is not as good as the new ridge regression's. This differs from the original case, where the two had a virtually indistinguishable MSE. This makes great deal of sense, though, as we generally assume that as we increase the number of variables in a model, the MSE will decrease. In the lasso model, variables are dropped, while in the ridge model, variables never truly drop out, making it sensical then that the MSE for the ridge model would be lower.

Let's take a quick look at how many variables are remaining in the lasso model at our optimal lambda.

```
out = glmnet(x_new, y_new, alpha=1, lambda=grid)
lasso.coef = predict(out, type = "coefficients", s = new.lasso.lam)[1:97,]
length(lasso.coef[lasso.coef != 0])
```

```
## [1] 15
```

As we see here, there are only 15 non-zero variables (including the intercept) in our lasso regression using our ideal lambda value.

Problem set 2

Chapter 6 exercises

Exercise 2

(a) Of the four options, option iii "Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance." is true for the lasso regression. A lasso model constrains regular least squares by a lambda value, which (1) biases the coefficients closer to 0 but simultaneously (2) reduces the variance of the model. Increases in MSE come when the reduction in variance outweighs the increase in bias, as stated by this option.

(b) The same option (iii) is true for ridge regression as well.

Exercise 3

(a) Option iv: the training RSS will steadily decrease. As s increases, our coefficients will not be as strictly bound. Our model, now more flexible, will therefore have a higher training RSS.

(b) Option ii: the test RSS will decrease initially, and then eventually start increasing in a U shape. As s increases, our model becomes more flexible. This will lead to an initial better fit, but will verge into overfitting if given too much leeway. The test RSS will therefore decrease initially, but increase after.

(c) Option iii: the variance will steadily increase. As s increases, the coefficients have less of a penalty placed on them. This increases their potential variance, and therefore the potential variance of the model.

(d) Option iv: the bias will steadily decrease. As we increase s, the coefficients have less of a penalty placed on them. This reduces the potential bias of these coefficients, as they'd then be able to approach the value which minimizes the sum of the squared residuals (therefore decreasing bias).

(e) Option v: the irreducible error will remain constant. We cannot control what the irreducible error is with a penalty term (or anything), so it should not depend on what our s equals.

Exercise 4

(a) Option iii: the training RSS will steadily increase. As lambda increases, our coefficients have more of a penalty, decreasing the flexibility of the model. The training RSS will of the model will therefore increase.

(b) Option ii: the test RSS will decrease initially, and then eventually start increasing in a U shape. Our model becomes less flexible as lambda increases. Therefore, the test RSS will decrease intitially (as there is less overfitting) but will then increase when the increase bias outweighs the lowered variance.

(c) Option iv: the variance will steadily decrease. As lambda increases, the coefficients of the model face more penalities, decreasing the variance of the model.

(d) Option iii: the bias will steadily increase. As lambda increases, we place more penalities on the model, which biases the coefficients toward 0 (thus increasing bias).

(e) Option v: the irreducible error will remain constant. We cannot control the irreducible error.

Exercise 6

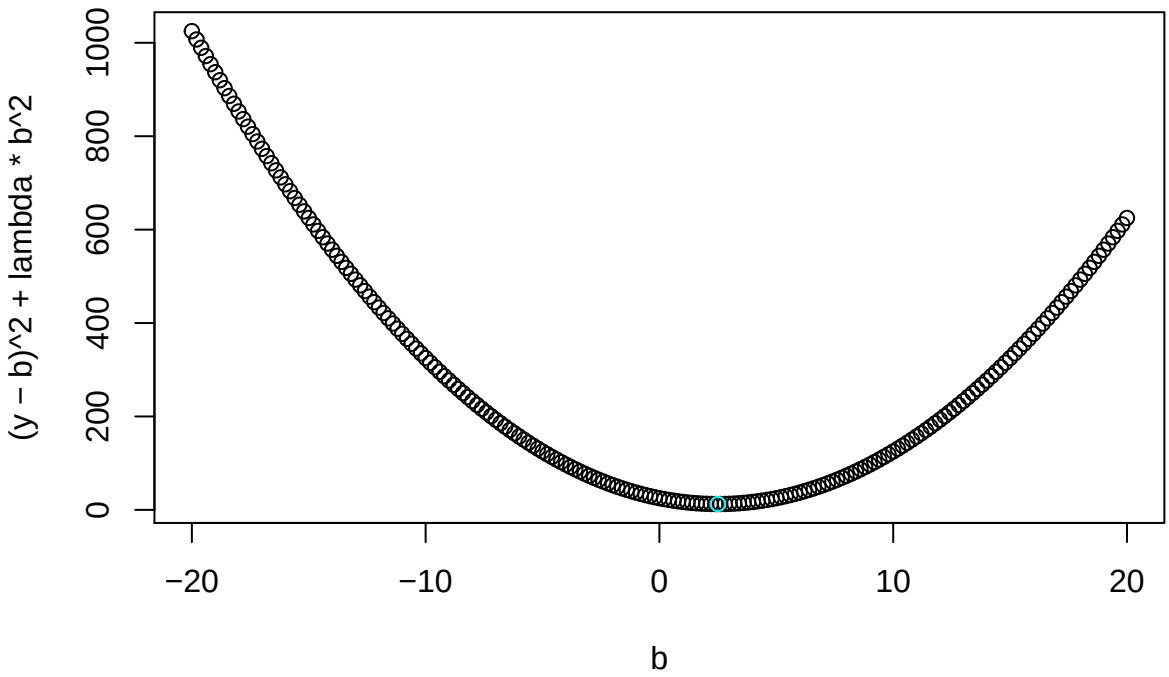
(a)

```
# Let's set up some lambda and y values, then generate a range of b values
lambda <- 1
y <- 5
b = seq(-20, 20, .2)

# Let's plot the equation 6.12 with p = 1
plot(b, (y-b)^2 + lambda*b^2)

# Equation 6.14
b_est <- y/(1+lambda)

# Let's plot the point given by Equation 6.14 to see if it minimizes Equation 6.12
points(b_est, (y-b_est)^2 + lambda*(b_est)^2, col="cyan")
```



As we can see, the blue point, which represents Equation 6.14, is at the minimum of Equation 6.12. In other words: Equation 6.12 is solved by Equation 6.14.

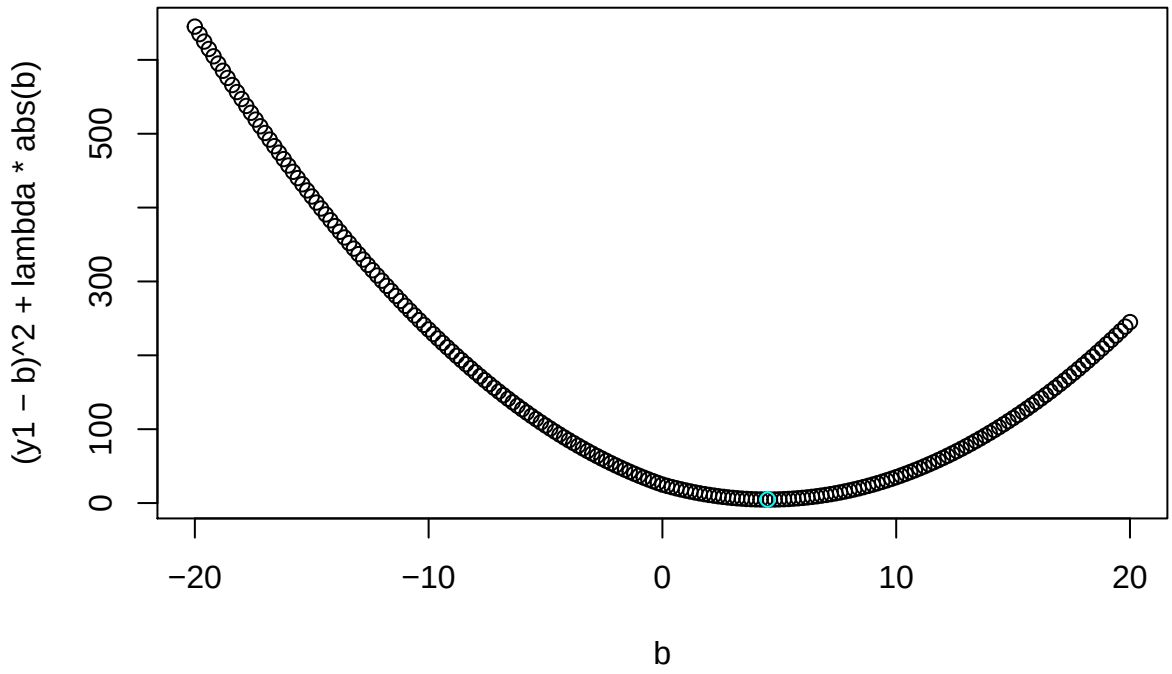
(b)

```
# Let's set up lambda and y values
lambda <- 1
y1 <- 5
b <- seq(-20, 20, 0.2)

# Let's plot Equation 6.13
p1 <- plot(b, (y1-b)^2 + lambda*abs(b))

# Equation 6.15 (since y > lambda/2)
b_est <- y1-lambda/2

# Let's plot the point given by Equation 6.15
points(b_est, (y-b_est)^2 + lambda*abs(b_est), col="cyan")
```



We can see that equation 6.15, shown in blue, solves for the minimum of the lasso regression model. In other words: Equation 6.13 is solved by Equation 6.15.