Chapter 6 in ISLR: Part 2 Regularization

Laura Kapitula

11/4/2020

This material on Ridge Regression and the Lasso in R comes from Chapter 6 of “Introduction to Statistical Learning with Applications in R” by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. I used code that did the lab in tidyverse format by Amelia McNamara and R. Jordan Crouser at Smith College. from <<http://www.science.smith.edu/~jcrouser/SDS293/labs/lab10.Rmd> > to implement the example in R.

## Ridge regression and Lasso

* The subset selection methods use least squares to fit a linear model that contains a subset of the predictors.

\*As an alternative, we can fit a model containing all predictors using a technique that constrains or regularizes the coefficient estimates, or equivalently, that shrinks the coefficient estimates towards zero.

* It may not be immediately obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance.

## Ridge Regression

Recall that the least squares fitting procedure estimates using values that minimize

In contrast, the ridge regression coefficient estimates are the values that minimize:

where $ $ is a tuning parameter, to be determined separately.

* As with least squares, ridge regression seeks coefficient estimates that fit the data well, by making the RSS small.
* However, the second term,

is called a shrinkage penalty, is small when are close to zero, and so it has the effect of shrinking the estimates of towards zero.

* The tuning parameter serves to control the relative impact of these two terms on the regression coefficient estimates.
* Selecting a good value for is critical and cross-validation is used for this.

library(ISLR)  
library(glmnet)  
library(dplyr)  
library(tidyr)

We will use the glmnet package in order to perform ridge regression and the lasso. The main function in this package is glmnet(), which can be used to fit ridge regression models, lasso models, and more. This function has slightly different syntax from other model-fitting functions that we have encountered thus far in this book. In particular, we must pass in an matrix as well as a vector, and we do not use the syntax.

Hitters = na.omit(Hitters)

We will now perform ridge regression and the lasso in order to predict Salary on the Hitters data. Let’s set up our data:

x = model.matrix(Salary~., Hitters)[,-1] # trim off the first column  
 # leaving only the predictors  
y = Hitters %>%  
 select(Salary) %>%  
 unlist() %>%  
 as.numeric()

The model.matrix() function is particularly useful for creating ; not only does it produce a matrix corresponding to the 19 predictors but it also automatically transforms any qualitative variables into dummy variables. The latter property is important because glmnet() can only take numerical, quantitative inputs.

The glmnet() function has an alpha argument that determines what type of model is fit. If alpha = 0 then a ridge regression model is fit, and if alpha = 1 then a lasso model is fit. We first fit a ridge regression model:

grid = 10^seq(10, -2, length = 100)  
ridge\_mod = glmnet(x, y, alpha = 0, lambda=grid)

By default the glmnet() function performs ridge regression for an automatically selected range of values. However, here we have chosen to implement the function over a grid of values ranging from to , essentially covering the full range of scenarios from the null model containing only the intercept, to the least squares fit.

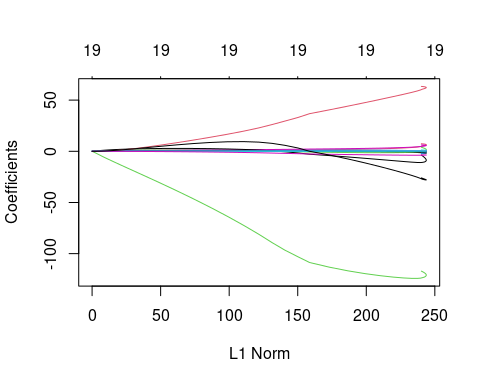
As we will see, we can also compute model fits for a particular value of that is not one of the original grid values. Note that by default, the glmnet() function standardizes the variables so that they are on the same scale. To turn off this default setting, use the argument standardize = FALSE.

### Ridge regression: scaling of predictors

* The standard least squares coefficient estimates are scale equivariant.
* In contrast, the ridge regression coefficient estimates can change substantially when multiplying a given predictor by a constant, due to the sum of squared coefficients term in the penalty part of the ridge regression objective function.
* Therefore, it is best to apply ridge regression after standardizing the predictors.

Associated with each value of is a vector of ridge regression coefficients, stored in a matrix that can be accessed by coef(). In this case, it is a matrix, with 20 rows (one for each predictor, plus an intercept) and 100 columns (one for each value of ).

plot(ridge\_mod)

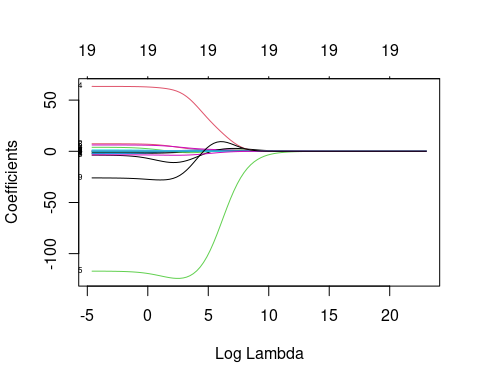


dim(coef(ridge\_mod))

## [1] 20 100

# Draw plot of coefficients

plot(ridge\_mod, xvar = "lambda", label = TRUE)

 We expect the coefficient estimates to be much smaller, in terms of norm, when a large value of is used, as compared to when a small value of is used. These are the coefficients when , along with their norm:

ridge\_mod$lambda[50] #Display 50th lambda value

## [1] 11497.57

coef(ridge\_mod)[,50] # Display coefficients associated with 50th lambda value

## (Intercept) AtBat Hits HmRun Runs   
## 407.356050200 0.036957182 0.138180344 0.524629976 0.230701523   
## RBI Walks Years CAtBat CHits   
## 0.239841459 0.289618741 1.107702929 0.003131815 0.011653637   
## CHmRun CRuns CRBI CWalks LeagueN   
## 0.087545670 0.023379882 0.024138320 0.025015421 0.085028114   
## DivisionW PutOuts Assists Errors NewLeagueN   
## -6.215440973 0.016482577 0.002612988 -0.020502690 0.301433531

sqrt(sum(coef(ridge\_mod)[-1,50]^2)) # Calculate l2 norm

## [1] 6.360612

In contrast, here are the coefficients when , along with their norm. Note the much larger norm of the coefficients associated with this smaller value of .

ridge\_mod$lambda[60] #Display 60th lambda value

## [1] 705.4802

coef(ridge\_mod)[,60] # Display coefficients associated with 60th lambda value

## (Intercept) AtBat Hits HmRun Runs RBI   
## 54.32519950 0.11211115 0.65622409 1.17980910 0.93769713 0.84718546   
## Walks Years CAtBat CHits CHmRun CRuns   
## 1.31987948 2.59640425 0.01083413 0.04674557 0.33777318 0.09355528   
## CRBI CWalks LeagueN DivisionW PutOuts Assists   
## 0.09780402 0.07189612 13.68370191 -54.65877750 0.11852289 0.01606037   
## Errors NewLeagueN   
## -0.70358655 8.61181213

sqrt(sum(coef(ridge\_mod)[-1,60]^2)) # Calculate l2 norm

## [1] 57.11001

We can use the predict() function for a number of purposes. For instance, we can obtain the ridge regression coefficients for a new value of , say 50:

predict(ridge\_mod, s = 50, type = "coefficients")[1:20,]

## (Intercept) AtBat Hits HmRun Runs   
## 4.876610e+01 -3.580999e-01 1.969359e+00 -1.278248e+00 1.145892e+00   
## RBI Walks Years CAtBat CHits   
## 8.038292e-01 2.716186e+00 -6.218319e+00 5.447837e-03 1.064895e-01   
## CHmRun CRuns CRBI CWalks LeagueN   
## 6.244860e-01 2.214985e-01 2.186914e-01 -1.500245e-01 4.592589e+01   
## DivisionW PutOuts Assists Errors NewLeagueN   
## -1.182011e+02 2.502322e-01 1.215665e-01 -3.278600e+00 -9.496680e+00

We now split the samples into a training set and a test set in order to estimate the test error of ridge regression and the lasso.

set.seed(1)  
  
train = Hitters %>%  
 sample\_frac(0.5)  
  
test = Hitters %>%  
 setdiff(train)  
  
x\_train = model.matrix(Salary~., train)[,-1]  
x\_test = model.matrix(Salary~., test)[,-1]  
  
y\_train = train %>%  
 select(Salary) %>%  
 unlist() %>%  
 as.numeric()  
  
y\_test = test %>%  
 select(Salary) %>%  
 unlist() %>%  
 as.numeric()

Next we fit a ridge regression model on the training set, and evaluate its MSE on the test set, using . Note the use of the predict() function again: this time we get predictions for a test set, by replacing type="coefficients" with the newx argument.

ridge\_mod = glmnet(x\_train, y\_train, alpha=0, lambda = grid, thresh = 1e-12)  
ridge\_pred = predict(ridge\_mod, s = 4, newx = x\_test)  
mean((ridge\_pred - y\_test)^2)

## [1] 139858.6

The test MSE is 101242.7. Note that if we had instead simply fit a model with just an intercept, we would have predicted each test observation using the mean of the training observations. In that case, we could compute the test set MSE like this:

mean((mean(y\_train) - y\_test)^2)

## [1] 224692.1

We could also get the same result by fitting a ridge regression model with a very large value of . Note that 1e10 means .

ridge\_pred = predict(ridge\_mod, s = 1e10, newx = x\_test)  
mean((ridge\_pred - y\_test)^2)

## [1] 224692.1

So fitting a ridge regression model with leads to a much lower test MSE than fitting a model with just an intercept. We now check whether there is any benefit to performing ridge regression with instead of just performing least squares regression. Recall that least squares is simply ridge regression with .

\* Note: In order for glmnet() to yield the **exact** least squares coefficients when , we use the argument exact=T when calling the predict() function. Otherwise, the predict() function will interpolate over the grid of values used in fitting the glmnet() model, yielding approximate results. Even when we use exact = T, there remains a slight discrepancy in the third decimal place between the output of glmnet() when and the output of lm(); this is due to numerical approximation on the part of glmnet().

ridge\_pred = predict(ridge\_mod, s = 0, newx = x\_test)  
mean((ridge\_pred - y\_test)^2)

## [1] 174060

lm(Salary~., data = train)

##   
## Call:  
## lm(formula = Salary ~ ., data = train)  
##   
## Coefficients:  
## (Intercept) AtBat Hits HmRun Runs RBI   
## 2.398e+02 -1.639e-03 -2.179e+00 6.337e+00 7.139e-01 8.735e-01   
## Walks Years CAtBat CHits CHmRun CRuns   
## 3.594e+00 -1.309e+01 -7.136e-01 3.316e+00 3.407e+00 -5.671e-01   
## CRBI CWalks LeagueN DivisionW PutOuts Assists   
## -7.525e-01 2.347e-01 1.322e+02 -1.346e+02 2.099e-01 6.229e-01   
## Errors NewLeagueN   
## -4.616e+00 -8.330e+01

predict(ridge\_mod, s = 0, type="coefficients")[1:20,]

## (Intercept) AtBat Hits HmRun Runs   
## 239.89368111 -0.01946204 -2.07305757 6.44254692 0.64610179   
## RBI Walks Years CAtBat CHits   
## 0.82179888 3.62448842 -13.28142313 -0.70314292 3.26064805   
## CHmRun CRuns CRBI CWalks LeagueN   
## 3.33170237 -0.54000590 -0.72015101 0.22582579 131.41324242   
## DivisionW PutOuts Assists Errors NewLeagueN   
## -134.76073238 0.20949301 0.61942855 -4.58545824 -82.35090554

It looks like we are indeed improving over regular least-squares! Side note: in general, if we want to fit a (unpenalized) least squares model, then we should use the lm() function, since that function provides more useful outputs, such as standard errors and -values for the coefficients.

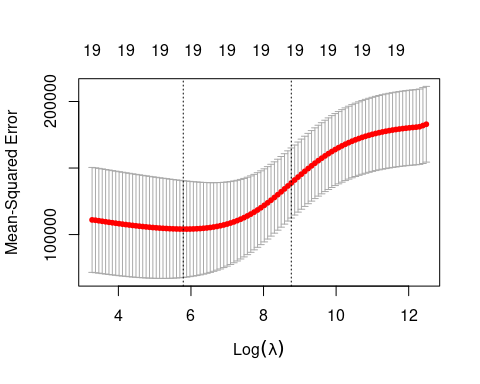
Instead of arbitrarily choosing , it would be better to use cross-validation to choose the tuning parameter . We can do this using the built-in cross-validation function, cv.glmnet(). By default, the function performs 10-fold cross-validation, though this can be changed using the argument folds. Note that we set a random seed first so our results will be reproducible, since the choice of the cross-validation folds is random.

set.seed(1)  
cv.out = cv.glmnet(x\_train, y\_train, alpha = 0) # Fit ridge regression model on training data  
bestlam = cv.out$lambda.min # Select lamda that minimizes training MSE  
bestlam

## [1] 326.1406

Therefore, we see that the value of that results in the smallest cross-validation error is 339.1845 We can also plot the MSE as a function of :

plot(cv.out) # Draw plot of training MSE as a function of lambda



What is the test MSE associated with this value of ?

ridge\_pred = predict(ridge\_mod, s = bestlam, newx = x\_test) # Use best lambda to predict test data  
mean((ridge\_pred - y\_test)^2) # Calculate test MSE

## [1] 140056.2

This represents a further improvement over the test MSE that we got using . Finally, we refit our ridge regression model on the full data set, using the value of chosen by cross-validation, and examine the coefficient estimates.

out = glmnet(x, y, alpha = 0) # Fit ridge regression model on full dataset  
predict(out, type = "coefficients", s = bestlam)[1:20,] # Display coefficients using lambda chosen by CV

## (Intercept) AtBat Hits HmRun Runs RBI   
## 15.44834992 0.07716945 0.85906253 0.60120338 1.06366687 0.87936073   
## Walks Years CAtBat CHits CHmRun CRuns   
## 1.62437580 1.35296285 0.01134998 0.05746377 0.40678422 0.11455696   
## CRBI CWalks LeagueN DivisionW PutOuts Assists   
## 0.12115916 0.05299953 22.08942756 -79.03490992 0.16618830 0.02941513   
## Errors NewLeagueN   
## -1.36075645 9.12528397

As expected, none of the coefficients are exactly zero - ridge regression does not perform variable selection!

## The Lasso

Ridge regression does have one obvious disadvantage: unlike subset selection, which will generally select models that involve just a subset of the variables, ridge regression will include all p predictors in the final model

* The Lasso is a relatively recent alternative to ridge regression that overcomes this disadvantage. The lasso coefficients, estimates are the values that minimize:

where $ $ is a tuning parameter, to be determined separately.

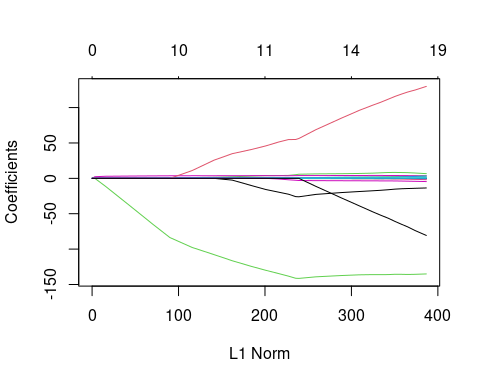
* As with least squares, Lasso seeks coefficient estimates that fit the data well, by making the RSS small.
* The second term,

is a shrinkage penalty, based on the norm. When we add up the squared terms it is a norm . For the Lasso, the can shrink the estimates of not just towards zero, but make them exactly equal to zero.

* However, in the case of the lasso, the penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter is sufficiently large.
* Hence, much like best subset selection, the lasso performs variable selection.
* We say that the lasso yields sparse models (does not use all predictors).
* Selecting a good value for is critical and cross-validation is used for this.

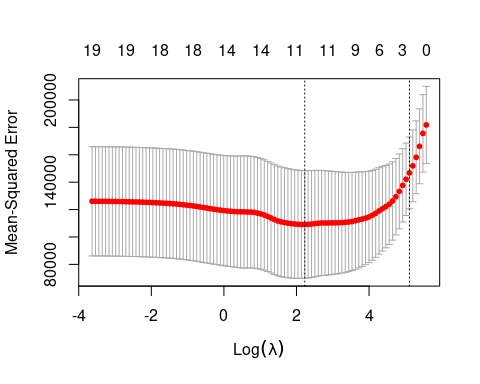
We saw that ridge regression with a wise choice of can outperform least squares as well as the null model on the Hitters data set. We now ask whether the lasso can yield either a more accurate or a more interpretable model than ridge regression. In order to fit a lasso model, we once again use the glmnet() function; however, this time we use the argument alpha=1. Other than that change, we proceed just as we did in fitting a ridge model:

lasso\_mod = glmnet(x\_train,   
 y\_train,   
 alpha = 1,   
 lambda = grid) # Fit lasso model on training data  
  
plot(lasso\_mod) # Draw plot of coefficients



Notice that in the coefficient plot that depending on the choice of tuning parameter, some of the coefficients are exactly equal to zero. We now perform cross-validation and compute the associated test error:

set.seed(1)  
cv.out = cv.glmnet(x\_train, y\_train, alpha = 1) # Fit lasso model on training data  
plot(cv.out) # Draw plot of training MSE as a function of lambda



bestlam = cv.out$lambda.min # Select lamda that minimizes training MSE  
lasso\_pred = predict(lasso\_mod, s = bestlam, newx = x\_test) # Use best lambda to predict test data  
mean((lasso\_pred - y\_test)^2) # Calculate test MSE

## [1] 143273

This is substantially lower than the test set MSE of the null model and of least squares, and very similar to the test MSE of ridge regression with chosen by cross-validation.

However, the lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse. Here we see that 12 of the 19 coefficient estimates are exactly zero:

out = glmnet(x, y, alpha = 1, lambda = grid) # Fit lasso model on full dataset  
lasso\_coef = predict(out, type = "coefficients", s = bestlam)[1:20,] # Display coefficients using lambda chosen by CV  
lasso\_coef

## (Intercept) AtBat Hits HmRun Runs   
## 1.27429897 -0.05490834 2.18012455 0.00000000 0.00000000   
## RBI Walks Years CAtBat CHits   
## 0.00000000 2.29189433 -0.33767315 0.00000000 0.00000000   
## CHmRun CRuns CRBI CWalks LeagueN   
## 0.02822467 0.21627609 0.41713051 0.00000000 20.28190194   
## DivisionW PutOuts Assists Errors NewLeagueN   
## -116.16524424 0.23751978 0.00000000 -0.85604181 0.00000000

Selecting only the predictors with non-zero coefficients, we see that the lasso model with chosen by cross-validation contains only seven variables:

lasso\_coef[lasso\_coef != 0] # Display only non-zero coefficients

## (Intercept) AtBat Hits Walks Years   
## 1.27429897 -0.05490834 2.18012455 2.29189433 -0.33767315   
## CHmRun CRuns CRBI LeagueN DivisionW   
## 0.02822467 0.21627609 0.41713051 20.28190194 -116.16524424   
## PutOuts Errors   
## 0.23751978 -0.85604181