BerkeleyHaas
Haas School of Business
University of California Berkeley

Big Data and Better Decisions
Spring 2018

Module 11: Lasso and Ridge Regression

1. INTRODUCTION

Recall the standard linear regression we've been using to describe the relationship between a response variable Y and a set of covariates X_n:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n + \epsilon$$

We typically use Ordinary Least Squares (OLS) to estimate this model. So far in this class, we've mostly used it as a tool for estimating treatment effects and making causal inferences. However, we can also use it for prediction! Many of the statistical learning techniques we'll cover (including KNN from last week) are non-linear, and this can be a more general and flexible way to fit data to a model. However, linear models are preferable when we want our underlying model to be interpretable, and often fit real-world data fairly well, even in comparison to more complex non-linear models. We can expand our toolkit from OLS to alternative fitting procedures, which will help us to improve **prediction accuracy** and **interpretability** even further.

We've established a tradeoff between the **bias** and **variance** of a predictive model that lead to predictive errors. With linear regression, we decrease bias with every variable we add to the regression. The more variables, the (weakly) higher the R squared. However, this comes at a cost of increasing the model variance—our model might do well on the training data, but if we try to use it to make predictions from the same variables in a different set of observations, the error rate will eventually start to go up as we add more variables. This is another example of the curse of dimensionality—too many predictors will result in overfitting and poor prediction. However, we can have the best of both worlds by shrinking or constraining the estimated coefficients (for example, by pushing the ones on predictors that don't actually matter to 0). This can lead to potentially large reductions in variance with little increase in bias. At the same time, removing irrelevant variables keeps the model from becoming unnecessarily complex, and makes it easier to interpret.

There are 3 main types of methods that go beyond OLS and accomplish the above goals with a linear model. The first is **subset selection**, which identifies a subset of predictors that we believe is related to the response, and fits the model using least squares, with this reduced set of predictors. **Shrinkage** fits a model with all predictors but shrinks the predictors toward 0 to reduce the variance—some may be estimated at exactly 0, which effectively selects a subset of predictors. **Dimension reduction** involves projecting the predictors into a subspace with fewer dimensions than the number of predictors, by computing linear combinations of the variables to match the target dimension. Here, we're going to focus the two best known methods that fall into the shrinkage category.

2. RIDGE AND LASSO REGRESSION

When we fit data using OLS, we estimate coefficients for each covariate by minimizing the **residual sum of squares** which is just an error term based on difference between the actual and predicted values from the resulting coefficients $\beta_0, \beta_1, \ldots., \beta_p$:

$$RSS = \sum_{i=1}^{n}(y_i - \hat{y_i})^2 = \sum_{i=1}^{n}\left(y_i - (\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij})\right)^2$$

Ridge regression works similarly, but measures success not only by the difference between the actual and predicted values, but also by the values of the coefficients themselves. For a ridge regression, the model is estimated by minimizing:

$$RSS + \lambda \sum_{j}^{p} \beta_j^2$$

which is the sum of the RSS and a **shrinkage penalty**. This means that the model is "penalized" for the non-zero values of the coefficients, having the effect of shrinking these coefficients toward 0. The value λ≥0 is called a tuning parameter and determines how much the error minimization takes into account the coefficient values. When λ=0 the ridge regression is the same as OLS, and as λ approaches infinity, the coefficients all approach 0. Thus, we obtain a different set of coefficients for each value of λ. Note that the shrinkage penalty applies to only the coefficients, and not the intercept β_0. Ridge regression's advantage over OLS goes back to the bias-variance tradeoff. With higher λ, the flexibility of the ridge regression fit decreases, leading to decreased variance but increased bias.

Though coefficients will be pushed toward 0 in the model produced by ridge regression, none of them will be exactly equal to 0. This is not a problem in terms of prediction accuracy, but if we have many predictor variables, it can make interpretation a challenge. **Lasso regression** addresses this disadvantage by shrinking some of the coefficients exactly to 0. For a lasso regression, the model is estimated by minimizing:

$$RSS + \lambda \sum_{j}^{p} |\beta_j|$$

Optimizing with the $\sum|\beta_j|$ penalty has the effect of forcing some of the estimated coefficients to be exactly for sufficiently large λ. In the next section, we'll learn how to select the optimal λ. Neither the ridge nor the lasso regression will universally dominate the other in terms of bias, variance, and MSE. Lass may be expected to dominate in settings where a small number of predictors have large coefficients and the rest of the predictors have small coefficients. Ridge may perform better when the outcome is a function of many predictors.

3. Selecting the Tuning Parameter

As discussed above, it's important to select the correct value of the tuning parameter λ to optimize the balance of bias and variance in our lasso or ridge model. We'll use a procedure called cross-validation to accomplish this. We'll discuss other applications of cross validation next week. **K-fold cross validation** involves randomly dividing the set of observation in to k groups of equal size. We'll use 10 groups in the R example below. The first group is treated as the test dataset, and the remaining 9 are treated as the training dataset to fit the model. We first fit the training dataset using a ridge (or lasso) regression with a given value for λ. The mean squared error is computed on the observations in the held-our test group, where $\hat{f}(x_i)$ is a predicted outcome value based on the

ridge regression estimated coefficients:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(x_i)\right)^2$$

This procedure is then repeated 10 (k) times total, with a different one of the 10 groups being held out as the test dataset each time. We then take the average of the 10 values for MSE, which gives us the **cross-validation error**:

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

We repeat this process for a grid of possible values of λ, each time calculating the cross validation error. We select the tuning parameter for which the cross-validation error is the smallest, and re-fit the model with all of the available observations and the selection of the tuning parameter. See the graph below from ISL, which shows the tuning parameter selection process using cross-validation with ridge regression, on a model with 3 predictor variables.

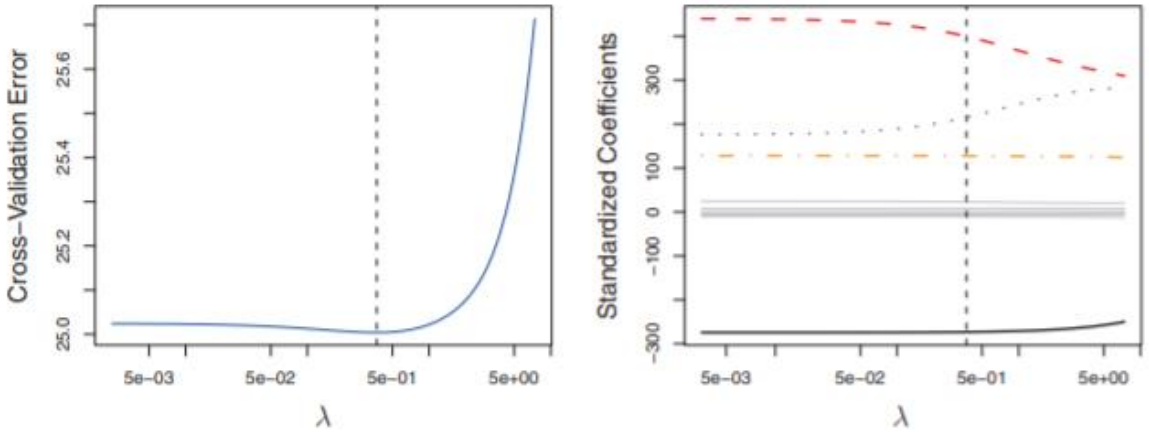

FIGURE 6.12. Left: *Cross-validation errors that result from applying ridge regression to the* `Credit` *data set with various value of* λ. Right: *The coefficient estimates as a function of* λ. *The vertical dashed lines indicate the value of* λ *selected by cross-validation.*

4. IMPLEMENTATION IN R

We will practice an example of lasso and ridge regression reproduced from section 6.6 of ISL, which uses the Hitters Data (loadable from JupyterHub). This data set is a Major League Baseball Dataset from the 1986 and 1987 seasons. The dataset has with 322 observations of major league players on the following 20 variables. Information on the individual variable names and the source is found in the appendix.

We will first read Hitters.csv and use `na.omit()` to remove missing data that has missing values. We also set our predictor variables vector X as all 19 variables in the dataset except y, the outcome variable Salary.

```
require (glmnet)
Hitters <- read.csv("Hitters.csv", header = TRUE)
Hitters <- na.omit(Hitters)
x=model.matrix(Salary~.,Hitters )[,-1]
y=Hitters$Salary
```

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 particular, we must pass in an x *matrix* as well as a y vector, and we do not use the y ~x syntax. The `model.matrix()` function is particularly useful for creating x ; 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.

Ridge Regression

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 λ values. However, here we have chosen to implement the function over a grid of values ranging from $\lambda = 10^{10}$ to $\lambda = 10^{-2}$, 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 λ 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`.

Associated with each value of λ is a vector of ridge regression coefficients, stored in a matrix that can be accessed by `coef()` . In this case, it is a 20x100 matrix, with 20 rows (one for each predictor, plus an intercept) and 100 columns (one for each value of λ).

```
dim(coef(ridge.mod))
```

We expect the coefficient estimates to be much smaller when a large value of λ is used, as compared to when a small value of λ is used. These are the coefficients when λ = 11,498, along with the value of $\sum_j^p \beta_j^2$ (ridge shrinkage penalty divided by λ):

```
ridge.mod$lambda[50]
coef(ridge.mod)[,50]
sqrt(sum(coef(ridge.mod)[-1,50]^2))
```

In contrast, here are the coefficients when λ = 705, along with the value of $\sum_j^p \beta_j^2$. Note the much larger value of $\sum_j^p \beta_j^2$ associated with this smaller value of λ.

```
ridge.mod$lambda[60]
coef(ridge.mod)[,60]
sqrt(sum(coef(ridge.mod)[-1,60]^2))
```

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 λ, say 50:

```
predict(ridge.mod,s=50,type="coefficients")[1:20 ,]
```

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. We randomly choose a subset of numbers between 1 and n; these can then be used as the indices for the training observations. We first set a random seed so that the results obtained will be reproducible.

```
set.seed(1)
train=sample(1:nrow(x), nrow(x)/2)
test=(-train)
y.test=y[test]
```

Next we want to fit a ridge regression model on the training set. Instead of arbitrarily choosing a value for λ, we can use cross-validation to choose the tuning parameter λ . We can do this using the built-in cross-validation function, `cv.glmnet()` . By default, the function `cv.glmnet()` performs ten-fold cross-validation, though this can be changed using the argument `nfolds`. Note that we set a random seed first so our results will be reproducible, since the choice of the cross-validation folds is random. You need to reset the seed each time you do a random sampling procedure.

```
set.seed(1)
cv.out=cv.glmnet(x[train ,],y[train],alpha =0)
plot(cv.out)
bestlam=cv.out$lambda.min
bestlam
```

Therefore, we see that the value of λ that results in the smallest cross validation error is 212. What is the test MSE associated with this value of λ?

```
ridge.pred=predict(ridge.mod,s=bestlam,newx=x[test ,])
mean((ridge.pred-y.test)^2)
```

Finally, we refit our ridge regression model on the full data set, using the value of λ chosen by cross-validation, and examine the coefficient estimates.

```
out=glmnet(x,y,alpha=0)
predict(out,type="coefficients",s=bestlam )[1:20 ,]
```

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

Lasso Regression

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)
plot(lasso.mod)
```

We can see from the coefficient plot that depending on the choice of tuning parameter, some of the coefficients will be 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)
plot(cv.out)
bestlam=cv.out$lambda.min
lasso.pred=predict(lasso.mod,s=bestlam,newx=x[test ,])
mean((lasso.pred-y.test)^2)
```

This is very similar to the test MSE of ridge regression with λ 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. So the lasso model with λ chosen by cross-validation contains only seven variables.

```
out=glmnet(x,y,alpha=1,lambda=grid)
lasso.coef=predict(out,type="coefficients",s=bestlam )[1:20 ,]
lasso.coef
```

Problem Set Questions 10.1

a) Fit a ridge regression model to the data from the Caravan auto insurance dataset using outcome Purchase and all of the other variables in the dataset as predictors.
   i. Load the data and necessary packages. Define the outcome using `y=as.factor(Insurance_Data$Purchase)` and x *matrix* using the `model.matrix()` function.
   ii. Split the data into a test and training dataset.
   iii. Fit a ridge regression to the test data over a grid of lambda values. You will need to use the argument `family="binomial"` in `glmnet` because the outcome is binary. You must indicate this argument each time you use glmnet or cv.glmnet.
   iv. Select the optimal lambda using cross-validation.
   v. Fit a ridge regression using the optimal λ and report the test error rate. To calculate the test error rate, you will need to re-format the test dataset outcome values (y.test in code from lab above) to numeric values using as.numeric().

b) Repeat steps iii-v above using a lasso regression. Also report the number of non-zero coefficients estimated. Compare the test error rates. Which is procedure do you think is preferable?

c) What do you notice about the predicted values for Purchase generated by Lasso and Ridge in this case? How do they compare to those generated from KNN?

5. BIBLIOGRAPHY/FURTHER READING

1. Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani (2013). An Introduction to Statistical Learning, Volume 112. Springer
2. Dataset Description - http://liacs.leidenuniv.nl/~puttenpwhvander/library/cc2000/data.html

6. APPENDIX

`AtBat`

Number of times at bat in 1986

`Hits`

Number of hits in 1986

`HmRun`

Number of home runs in 1986

`Runs`

Number of runs in 1986

`RBI`

Number of runs batted in in 1986

`Walks`

Number of walks in 1986

`Years`

Number of years in the major leagues

`CAtBat`

Number of times at bat during his career

`CHits`

Number of hits during his career

`CHmRun`

Number of home runs during his career

`CRuns`

Number of runs during his career

`CRBI`

Number of runs batted in during his career

`CWalks`

Number of walks during his career

`League`

A factor with levels `A` and `N` indicating player's league at the end of 1986

`Division`

A factor with levels `E` and `W` indicating player's division at the end of 1986

`PutOuts`

Number of put outs in 1986

`Assists`

Number of assists in 1986

`Errors`

Number of errors in 1986

`Salary`

1987 annual salary on opening day in thousands of dollars

`NewLeague`

A factor with levels `A` and `N` indicating player's league at the beginning of 1987

Source

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. This is part of the data that was used in the 1988 ASA Graphics Section Poster Session. The salary data were originally from Sports Illustrated, April 20, 1987. The 1986 and career statistics were obtained from The 1987 Baseball Encyclopedia Update published by Collier Books, Macmillan Publishing Company, New York.