Assignment #5

Lindsay Brock, Gustavo Esparza, and Brian Schetzsle

October 3, 2019

Problem #1: Original Code for ISLR Algorithms 6.1, 6.2, and 6.3

Algorithm 6.1: Best Subset Selection

To perform *best subset selection*, a least squares regression needs to be fit for each possible combination of the p predictors, then each model will need to be compared to identify which is *best.* This is done by beginning with a *null* model that contains no predictors, that only predicts the sample mean for each observation. Next, for $k = 1, \ldots, p$, fit all $\binom{p}{k}$ models that contain exactly k predictors and pick the best among this set of models (call it M_k), that which has the smallest RSS. Finally, select the best model from $M_0, \ldots, M_p$ using some cross-validated measure of error.

Algorithm 6.2: Forward Stepwise Selection

To perform *forward stepwise selection*, again begin with a model containing no predictors (M_0) and add them one at a time until all of the predictors have been added. Specifically, for $k = 0, \ldots, p-1$ create all $p-k$ models that augment the predictors in M_k, adding one additional predictor. Then choose the best of this subset of models (call it M_{k+1}), that with the smallest RSS. Finally, test to see which is best of the $M_0, \ldots, M_p$ models, as done in the previous method. This method is computationally less expensive than best subset selection.

Algorithm 6.3: Backward Stepwise Selection

This method is similar to forward stepwise selection but essentially works in the opposite direction. To perform *backward stepwise selection*, start with the saturated model containing all p predictors (M_p) and one at a time remove the least useful predictor. Specifically, for $k = p, p-1, \ldots, 1$ create all k models that contain all but one of the predictors in M_k which will total to $k-1$ predictors. Then, choose the best of these k models using the same RSS criteria and call it M_{k-1}. Finally, select the best model from $M_0, \ldots, M_p$ using some cross-validated measure of error.

Original Code for Best, Forward, and Backward SS

All three methods are performed in the "best_model" function. It begins by generating data where Y is a random function of X1, X2, X3, and X4. Squared and cubed predictors were also added to the data to allow the algorithm the chance to overfit. The library "rlist" is used to sort the list of candidate models based on R^2 and adjusted R^2.

```r
library(rlist)

X1 = rnorm(100,0,5)
X2 = rpois(100,5)
X3 = rbinom(100,100,0.3)
X4 = rnbinom(100,10,0.3)

X = cbind(X1,X2,X3,X4,X1^2,X2^2,X3^2,X4^2,X1^3,X2^3,X3^3,X4^3)
beta = matrix(c(runif(4,-5,5),rep(0,8)),nrow=1)
Y = X %*% t(beta) + rnorm(100,0,5)
```

```r
best_model = function(Y, X, method="best subset"){
  #get the number of predictors p
  p = dim(X)[2]
  #create a vector to hold the final (p+1) candidate models
  models = vector(mode="list", length=p+1)
  #the first candidate model is always the null model with no parameters
  models[[1]]$model = summary(lm(Y ~ rep(1,length(Y))))
  models[[1]]$predictors = vector()

  if(method=="forward stepwise"){
    for (k in 0:(p-1)){
      #create a vector to hold all the models that have one more predictor
      #than previous iteration
      potential_models = vector(mode="list", length=p-k)
      #get all the remaining predictors that are not in the previous best model
      potential_predictors = setdiff(seq(1:p), models[[k+1]]$predictors)
      for (i in 1:(p-k)){
        #consider all models that have the previous best model's predictors plus one more
        temp_predictors = union(models[[k+1]]$predictors, potential_predictors[i])
        potential_models[[i]]$model = summary(lm(Y ~ X[,temp_predictors]))
        potential_models[[i]]$predictors = temp_predictors
      }
      #get the model with the highest r.squared and save it for later comparison
      models[[k+2]] = list.sort(potential_models, -model$r.squared)[[1]]
    }
  }

  if(method=="backward stepwise"){
    #populate the saturated model at the end of the candidate model list
    models[[p+1]]$model = summary(lm(Y ~ X))
    models[[p+1]]$predictors = seq(1:p)
    #only go down to k=2 because the null model at k=1 is already in the list
    for (k in p:2){
      potential_models = vector(mode="list", length=k)
      for (i in 1:k){
        #remove each of the previous best model's predictors
        temp_predictors = models[[k+1]]$predictors[-i]
        potential_models[[i]]$model = summary(lm(Y ~X[,temp_predictors]))
        potential_models[[i]]$predictors = temp_predictors
      }
      #get the model with the highest r.squared and save it for later comparison
      models[[k]] = list.sort(potential_models, -model$r.squared)[[1]]
    }
  }

  else{
    for (k in 1:p){
      potential_models = vector(mode="list", length=choose(p,k))
      #get all possible combinations of k predictors
      potential_predictors = combn(p,k, simplify=FALSE)
      for (i in 1:length(potential_predictors)){
        temp_predictors = potential_predictors[[i]]
        potential_models[[i]]$model = summary(lm(Y ~ X[,temp_predictors]))
        potential_models[[i]]$predictors = temp_predictors
      }
      #get the model with the highest r.squared and save it for later comparison
      models[[k+1]] = list.sort(potential_models, -model$r.squared)[[1]]
    }
  }
  #find the candidate model with the largest adjusted r squared and return its predictors
  return(sort(list.sort(models, -model$adj.r.squared)[[1]]$predictors))
}

best = best_model(Y,X,method = "best subset")
forward = best_model(Y,X,method = "forward stepwise")
backward = best_model(Y,X,method = "backward stepwise")
```

Table 1: Variables for Best Model

Best	1	2	4	5	6	7
Forward	1	2	4	5	6	7
Backward	1	2	4	5	6	7

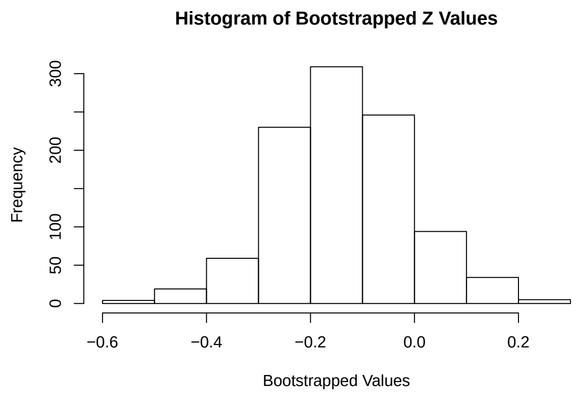
After running our algorithms, the best model found using each of the three methods will include the column variables represented in the table above.

Problem #2: Parametric and Nonparametric Bootstrap

a) Use bootstrap to find distribution of Fisher's z-transformation and 95% CI

$$z = \frac{1}{2}ln\left(\frac{1+r}{1-r}\right) \qquad where \qquad r = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

```
setwd("C:/Users/linds/OneDrive/Desktop/MATH 533/HW5")
pairs = read.csv("pair.txt",header = F)

set.seed(533)
z_bs = rep(0,1000)
for (i in 1:1000){
  pairs_bs = pairs[sample(nrow(pairs),replace=TRUE),]

  r = cor(pairs_bs[,1],pairs_bs[,2])
  z_bs[i] = .5*log( (1+r)/(1-r))
}

#Normal Assumption
mean = mean(z_bs)
se = sd(z_bs)
CI_normal_a = c(mean - 1.96*se, mean + 1.96*se)

#Percentile Method
CI_percentile_a = c(quantile(z_bs,.025), quantile(z_bs,.975))

hist(z_bs, main = "Histogram of Bootstrapped Z Values", xlab = "Bootstrapped Values")
```



Above is a histogram of the bootstrapped z values using data from the Pairs data set. It centers around a mean of -0.1364196 with the following confidence intervals,

	Confidence Interval
Normal Assumption	(-0.385, 0.113)
Percentile Method	(-0.395, 0.133)

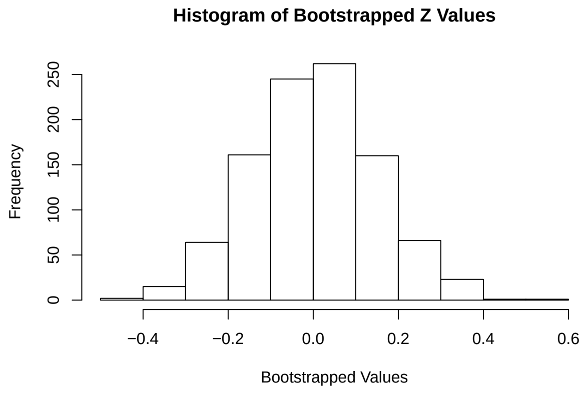
b) Repeat Part A but with data from Bivariate Normal

$$(x, y)^t \sim N_2(\mu^t, \Sigma) \qquad where \qquad \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$$

The mean vector μ of size 2 is unknown.

Using sample means for Bivariate Normal μ^T:

```r
library(MASS)
set.seed(533)
sigma = matrix(c(2,0,0,4),2,2)
mu1 = mean(pairs[,1])
mu2 = mean(pairs[,2])

for (i in 1:1000){
  pairs_bs = mvrnorm(50,c(mu1,mu2),sigma)
  r = cor(pairs_bs[,1],pairs_bs[,2])
  z_bs[i] = .5*log( (1+r)/(1-r))
}

#Normal Assumption
mean = mean(z_bs)
se = sd(z_bs)
CI_normal_b = c(mean - 1.96*se, mean + 1.96*se)

#Percentile Method
CI_percentile_b = c(quantile(z_bs,.025), quantile(z_bs,.975))

hist(z_bs, main = "Histogram of Bootstrapped Z Values", xlab = "Bootstrapped Values")
```



Above is a histogram of the bootstrapped z values using data generated from the Bivariate Normal distribution. It centers around a mean of 0.0027089 with the following confidence intervals,

	Confidence Interval
Normal Assumption	(-0.283, 0.288)
Percentile Method	(-0.276, 0.297)

c) Comparison of CIs

Formula (11.13) from Efron and Hastie:

$$C^{\phi}(\hat{\phi}) = \hat{\phi} \pm 1.96 \frac{1}{\sqrt{n-3}}$$

```
r = cor(pairs[,1],pairs[,2])
z = .5*log( (1+r)/(1-r))
se = 1/sqrt(50-3)

CI_EH = c(z-1.96*se, z+1.96*se)
```

	Confidence Interval
Normal - Pairs	(-0.385, 0.113)
Percentile - Pairs	(-0.395, 0.133)
Normal - B.N.	(-0.283, 0.288)
Percentile - B.N.	(-0.276, 0.297)
Efron and Hastie	(-0.412, 0.159)

Using the table above to compare the CI found using formula (11.13) from Efron and Hastie to those found in Parts A & B, all intervals have noticeable differences. The theoretical confidence interval's lower bound is lower than the other four intervals. It also seems to be wider than the intervals for Part A and similar in size to the intervals from Part B, just lower in value.

Section 6.2: Shrinkage Methods

In the previous section of this chapter, subset selection methods were discussed such as Best Subset Selection, Forward and Backward Stepwise Selection, and hybrid approaches, some of which are seen in the code in Problem #1. They utilize least squares to fit a linear model that contains a subset of predictors. In this section we will be discussing an alternative method of fitting a model, which includes all p predictors, by *constraining* or *regularizing* the coefficient estimates, which in turn *shrinks* them towards a value of zero. This method is useful because in shrinking the coefficient estimates, the variance is significantly reduced which is beneficial for any statistical analysis. This method and the explanation for its variance reduction is detailed with a discussion of two of the most widely used techniques today, *ridge regression* and the *lasso*.

In our summary for Chapter 3, fitting a model using least squares estimates the coefficients, $\beta_0, \ldots, \beta_p$, by using those that minimize,

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

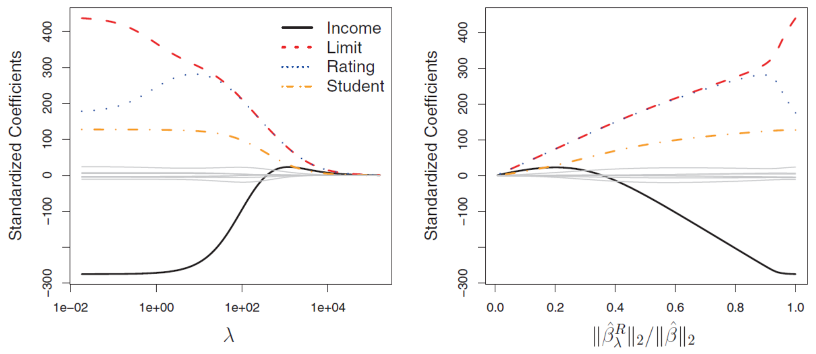
This new method of *ridge regression* is almost the same in that they continue to find the coefficient estimates, this time denoted by $\hat{\beta}^R$, that minimize a similar equation except for an added *shrinkage penalty.*

$$RSS + \lambda \sum_{j=1}^{p} \beta_j^2$$

As the β_j is squared in this term, it is small when $\beta_1, \ldots, \beta_p$ are close to zero which in turn *shrinks* their estimates towards zero as our goal is to minimize the overall equation. Notice it does not include β_0. This is because the intercept is only a measure of the mean value of the response when all other variables are zero. If we assume that each variable has a mean of zero before ridge regression, we can estimate the intercept using the equation for $\bar{y} = \sum_{i=1}^{n} y_i / n$.

The λ, or *turning parameter*, controls the impact the shrinkage penalty has on the overall equation, i.e. if it is 0 the penalty has no effect but as λ gets larger it continues to have a larger impact therefore shrinking the minimized estimates towards zero. It is also important to note that a different set of estimates for $\hat{\beta}^R$ are created for each λ. Selecting the appropriate λ is discussed later in this section.

The section continues to reinforce the concepts above with an example using data called Credit, which includes the variables *income, limit, rating*, and *student.* It begins with a plot (below left) of different λ values against the standardized coefficients (the definition of standardized is discussed later) to illustrate how the turning parameter effects the coefficient estimates. It is shown that when λ is small the estimates are the same as those found using the least squares method. When λ is large, the estimates become extremely close to zero (this model is called the *null model* as it would not contain any predictors). The trend is clear from the plot that as λ increases, the coefficient estimates shrink.



The example goes on to describe and plot (above right) the amount that the ridge regression coefficients have been shrunken towards zero, using the l_2 norm, with the shrinkage amount on the x-axis and standardized coefficients on the y-axis, and a line plotted for the estimate coefficient of each variable. The equation for the shrinkage amount is $||\hat{\beta}^R_\lambda||_2/||\hat{\beta}||_2$, where $||\hat{\beta}||_2 = \sqrt{\sum_{j=1}^{p} \beta_j^2}$ (l_2 norm or the distance of β to zero) and $\hat{\beta}$ is a vector of the coefficient estimates using least squares. As λ increases and the estimates shrink, the l_2 norm for the ridge regression estimates decreases (numerator) which causes the entire shrinkage amount to decrease. As the shrinkage amount is a proportion of the ridge estimates over the least squares estimates, the values will be between 0 (when $\lambda = \infty$ because the estimates will be close to zero) and 1 (when $\lambda = 0$ because the estimates will be the same as those found using least squares). This is confirmed by the graph where the plotted values of each coefficient estimate move away from the zero line on the y-axis as the proportion of distance increases along the x-axis. It may be helpful to envision this graph flipped with the x-axis values ranging from 1 to zero as the values would actually decrease, since they are the proportion of the distance the estimates are away from zero as λ increases.

The coefficient estimates found using least squares are *scale equivalent*, which means if a vector of a predictor variable is multiplied by a constant it does not significantly affect the value of the coefficient. The coefficient will simply be divided by the constant (scaled) to account for the change. Ridge regression on the other hand, does not follow that same trend because of the shrinkage penalty term. Now, $X_j\hat{\beta}^R_j$ not only depends on the coefficient but also the sum of squared coefficients, λ, and sometimes even the scaling of the other predictors, all of which can't just simply be divided by the constant. Because of this, it is important to *standardize the predictors* before applying ridge regression to ensure they are all on the same scale, using the following formula,

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}}$$

Note: Denominator = estimated standard deviation of the j^{th} predictor. The new standardized predictors will all have a standard deviation of 1.

Next, the section answers the question of why ridge regression improves over least squares and that is because of the *bias-variance trade-off.* When λ increases, increasing the effect of the shrinkage penalty term, the variance decreases because of the decrease in the flexibility of the ridge regression fit. With the shrinkage of the coefficients and this decreased variance, the coefficients can be severely underestimated which results in a substantial increase in bias, hence the trade-off.

When the response and predictors are considered to have a linear relationship, the estimates found using least squares will have low bias but high variance which means a small change in training data can have a major impact on the values of the estimates. In the context of the number of predictors, when that number is almost the same as the number of observations the estimates will be extremely variable meaning a high variance. When the number of predictors is greater than the number of observations, the estimates do not even have a unique solution. In these cases, ridge regression would be an appropriate solution as it decreases variance, even though bias may be increased. An additional advantage is that it is computationally easier when compared to best subset selection as it would not need to search through 2^p models, even when calculating for multiple λs.

The section goes on to discuss the next variance reduction method, the *lasso.* Unlike ridge regression which includes all predictor variables in the model even if that number is quite large, lasso is a more recent approach that follows similar rules but instead of bringing some of the estimated coefficients close to zero, actually brings them to zero, effectively eliminating that particular variable from the model. Lasso finds the estimated coefficients ($\hat{\beta}^L_\lambda$) by minimizing the following equation,

$$RSS + \lambda \sum_{j=1}^{p} |\beta_j|$$

As you can see the penalty term is different than that of ridge regression as the l_2 norm is replaced with the l_1 , $||\beta||_1 = \sum |\beta_j|$. This term functions much in the same way as it does in ridge with respect to λ, but because of the l_1 norm, forces some of the coefficient estimates to zero when λ is sufficiently large. Similarly to ridge, when $\lambda = 0$ the least squares model is produced, and when λ is sufficiently large the *null* model with all estimated coefficients equal to zero is produced. It is what happens in between those two extremes, the range of λs where estimated coefficients start to become zero instead of just close to it, that sets lasso apart from ridge. In essence this is what we call *variable selection* and selecting a subset of variables in this way gives us a *sparse* model.

It should also be noted that lasso coefficient estimates can solve,

$$\underset{\beta}{min} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad subject \quad to \quad \sum_{j=1}^{p} |\beta_j| \leq s \tag{6.8}$$

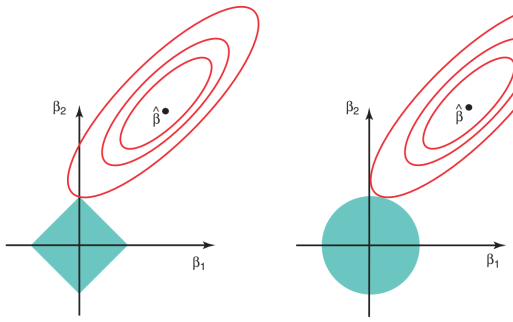
which means that for every value of λ, there is a region of s that will give the same coefficient estimates as the minimized $RSS + \lambda \sum_{j=1}^{p} |\beta_j|$.

Similarly, ridge regression coefficient estimates can solve,

$$\underset{\beta}{min} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad subject \quad to \quad \sum_{j=1}^{p} \beta_j^2 \leq s \tag{6.9}$$

with some region of s that will give the same coefficients as the minimized $RSS + \lambda \sum_{j=1}^{p} \beta_j^2$.

To illustrate this concept let's use an example of $p = 2$. For lasso, when $p = 2$ in the support above, $|\beta_1| + |\beta_2| \leq s$. The coefficients that minimize the RSS will lie within the diamond that is defined by that equation. For ridge, $\beta_1^2 + \beta_2^2 \leq s$ (a circle) and the coefficients that minimize the RSS will lie within that circle. This s can also be understood as a type of *budget* for how large each equation (and in turn each estimate) can really be. When the budget is large the estimates can also be large, even large enough to include the estimations from least squares in which case they will be the final solution. If the budget is small, then the estimates must be small to avoid going over.



Now going back to the subject of variable selection in lasso, you may be wondering how the coefficients actually become zero instead of just close to it as in ridge. To explain we will continue with our example of $p = 2$, using the two plots above, the left of lasso and the right of ridge. The x-axis represents β_1 and y-axis, β_2. The points on the red ellipses around the least squares coefficient estimate ($\hat{\beta}$) represent the shared RSS for different β_1s and β_2s and as the ellipses get larger, so do the RSS values. The blue circle and square represent the budget of s. The equations (6.8) and (6.9), which minimize the RSS within the constraints, tell us that the coefficient estimates are in fact the first point where the red ellipse touches the edge of the budget. As the lasso budget is square and has sharp points, the ellipse will often intersect on an axis making one of the βs equal to zero. Since the budget for ridge is circular with no sharp points, the intersection has an extremely slim chance of happening directly on an axis making the estimates non-zero. The same reasoning is used for a p of higher dimensions, with lasso continuing to have sharp edges and ridge, round.

Now that lasso and ridge regression have been thoroughly explained, the section continues with a more in depth comparison of the two methods. The first advantage lasso has over ridge is that it produces simpler models using a smaller subset of predictors which is easier to interpret. But speaking to which model leads to better prediction accuracy, neither model universally outperforms the other. The text reinforces this claim with plots, specifically of how variance and bias behave as λ increases, as well as a cross-validated MSE comparison. Even though one method is not empirically superior to the other, they do each have their own best case scenarios. Lasso performs better when there are fewer predictors with larger coefficients and the rest become zero or close to it. On the other hand, ridge does best when there are many predictors that all have roughly equal coefficients. As always, our tried and true cross-validation can be used to decide on the best method per situation.

As stated previously, when it is realized that estimates found using least squares have high variance, lasso can be used to reduce that variance with a trade off of a small increase in bias which in turn provides more accurate predictions. It can also perform variable selection, which ridge cannot.

Next, a special case of lasso and ridge regression is discussed, where $n = p$ and $\boldsymbol{X}$ is an identity matrix. It is also assumed we are performing regression without an intercept. With this information and using least squares we find $\beta_1, \ldots, \beta_p$ that minimizes the equation $\sum_{j=1}^{p}(y_j - \beta_j)^2$ and find the solution to be $\hat{\beta}_j = y_j$.

Again using the given information and ridge regression we minimize the equation below to find the values of $\beta_1, \ldots, \beta_p$.

$$\sum_{j=1}^{p}(y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

where the estimates are calculated to be,

$$\hat{\beta}_j^R = y_j/(1 + \lambda)$$

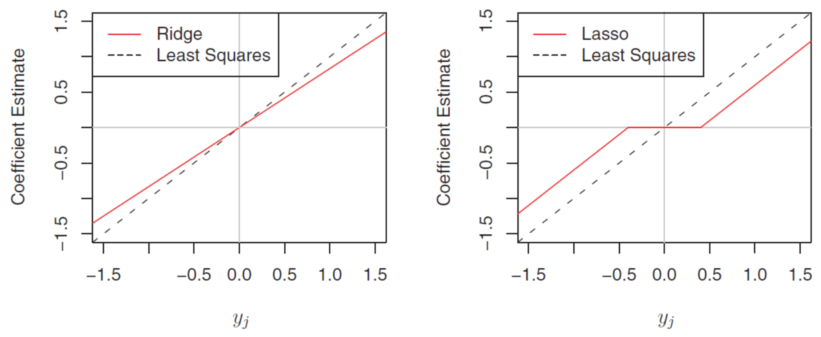
And similarly for lasso,

$$\sum_{j=1}^{p}(y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

with estimates,

$$\hat{\beta}_j^L = \begin{cases} y_j - \lambda/2 & if \quad y_j > \lambda/2 \\ y_j + \lambda/2 & if \quad y_j < -\lambda/2 \\ 0 & if \quad |y_j| \leq \lambda/2 \end{cases}$$

Below are the plots of the estimates above, $\hat{\beta}_j^R$ and $\hat{\beta}_j^L$, against their response variable values. The black dotted line represents the estimated coefficients from least squares and the red lines represent those from ridge regression and lasso, respectively. They are provided in this section to help explain how the types of shrinkage for both methods are performed and why they are different.



The plot on the left represents the coefficient estimates for ridge regression. It shows that there is a linear relationship between the estimates and the response (red line) meaning the least squares estimates are shrunken consistently by the same proportion. The plot on the right represents the coefficient estimates for lasso and it is clearly different from that of ridge. Following the conditions above for $\hat{\beta}_j^L$, from -1.5 to around -0.5 where $y_j < -\lambda/2$ the estimates (red line) are shrunken by adding the proportion $\lambda/2$ to the least squares estimate, from -0.5 to 0.5 where $|y_j| \leq \lambda/2$ they shrink entirely to zero, and then from 0.5 on, where $y_j > \lambda/2$, subtract $\lambda/2$ from the least squares estimate. This specific type of shrinkage is called *soft-thresholding*. Even though it is more complicated to explain, this idea that ridge shrinks all of the coefficients by a consistent proportion while lasso shrinks the coefficients towards zero by a similar amount also shrinking those that are extremely small to zero, can be extended to a general data set that we might normally expect to see.

Moving on, this section brings forth the topic of the Bayesian interpretation for ridge regression and lasso. To set the scene for both methods, we know well that the basis for anything Bayesian involves a *prior* distribution and when it comes to regression under this viewpoint, the vector of coefficients ($\beta = (\beta_0, \dots, \beta_p)^T$) will have a prior distribution, $p(\beta)$. The *posterior distribution* is below,

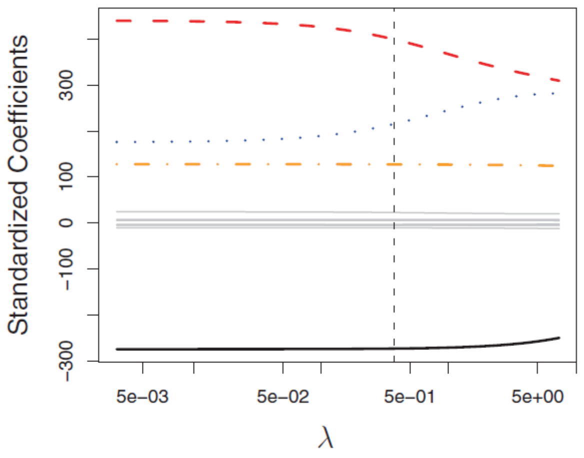
$$p(\beta|\boldsymbol{X},\boldsymbol{Y}) \propto f(\boldsymbol{Y}|\boldsymbol{X},\beta)p(\beta|\boldsymbol{X}) = f(\boldsymbol{Y}|\boldsymbol{X},\beta)p(\beta)$$

where $f(\boldsymbol{Y}|\boldsymbol{X},\beta)$ is the likelihood and $\boldsymbol{X} = (X_1, \dots, X_p)$. Also, $p(\beta|\boldsymbol{X},\boldsymbol{Y})$ is proportional to the rest of Bayes' theorem that is not included and $p(\beta|\boldsymbol{X}) = p(\beta)$ under the assumption that $\boldsymbol{X}$ is fixed.

Using our usual linear model equation $Y = \beta_0 + X_1\beta_1 + \cdots + X_p\beta_p + \epsilon$, with errors that are independent and drawn from a Normal distribution, we assume that the prior, $p(\beta)$, equals $\prod_{j=1}^{p} g(\beta_j)$ where g is a density function, either Gaussian or double-exponential. With a Gaussian g with a mean of zero and standard deviation a function of λ, we find the *posterior mode*, or most likely value for β given the data, is actually given by the estimates found using ridge regression. They also happen to be the posterior mean. With a double-exponential (Laplace) g with mean zero and scale parameter a function of λ, the posterior mode for β are the estimates found using lasso but in this case they are not the posterior mean as before.

Finally, the section closes with a discussion of the final piece in building our ridge regression and lasso models, how to select the most appropriate turning parameter λ. You have probably already guessed, it involves cross-validation! A range of λs are chosen, your choice of cross-validation is used to calculate the error for each value, and the λ with the smallest error is selected. The final model is then fit using the data and selected λ. Voila! You now have a complete understanding and model for both ridge regression and lasso.

Side Note: The graph below displays the different values of λ against the standardized coefficient values. The red and orange dotted lines represent the two predictors in this data set that are related to the response while the grey lines represent those that are not. These are referred to *signal* and *noise* variables. From the plot we can see that the appropriate λ denoted by the dotted vertical line, gives larger coefficients to the *signal* variables while giving the *noise* variables coefficients close to zero. This is useful as it can correctly identify important signal variables out of a high dimensional data set.



Problem #4: Questions 5, 6, and 9 from ISLR

Question #5

Using the Default data set we will estimate the test error of a logistic regression model, fit to predict the probability of *default* using *income* and *balance*, using the validation set approach.

a) Logistic Regression Model

```r
library(ISLR)
defdat = Default
attach(defdat)

model_5a = glm(default~income+balance, data = defdat, family = binomial)
summary(model_5a)
```

```

Call:
glm(formula = default ~ income + balance, family = binomial,
    data = defdat)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4725  -0.1444  -0.0574  -0.0211   3.7245

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2920.6  on 9999  degrees of freedom
```

```
Residual deviance: 1579.0  on 9997  degrees of freedom
AIC: 1585

Number of Fisher Scoring iterations: 8
```

Above is a summary of the logistic regression model with *default* as the response and *income* and *balance* as predictors. It includes the estimate and standard deviation of each variable's coefficient.

b) Estimate the test error of the model above using the validation set approach

```r
#Testing and training data
n = length(income)
set.seed(533)
index = sample(1:n, n*.10, replace = F)
test = defdat[index,]
train = defdat[-index,]

#Training model using training data
model_5b_error = glm(default~income+balance, data = train, family = binomial)

#Predictions using testing data
pred_5b = ifelse(predict.glm(model_5b_error, newdata = test, type = "response")>0.5, "Yes", "No")

#Test error using validation approach
test_error_5b = mean(test[,1] != pred_5b)
```

Using the validation approach, the test error of the model from Part A is 0.024.

c) Repeat of Part B with three different splits of test/train data

```r
test_error_5c = matrix(rep(0,3), 3)

for(i in 1:3){
  index = sample(1:n, n*.10, replace = F)
  test = defdat[index,]
  train = defdat[-index,]

  model_error= glm(default~income+balance, data = train, family = binomial)
  pred = ifelse(predict.glm(model_error, newdata = test, type = "response")>0.5, "Yes", "No")
  test_error_5c[i,] = mean(test[,1] != pred)
}
```

	Test Error
Split 1	0.034
Split 2	0.025
Split 3	0.028

Above are the test error results using the validation method with three unique testing and training splits. It appears that the models fit well as the errors are low and provide similar results for all three separate validation iterations. The test error rates are close for each split of the test/train data as well as the test error from Part B.

d) Logistic regression to predict *default* using *income*, *balance*, and dummy variable for *student*

```r
defdat$student = ifelse(student == "Yes", 1, 0)

test_error_5d = matrix(rep(0,3), 3)

for(i in 1:3){
  index = sample(1:n, n*.10, replace = F)
  test = defdat[index,]
  train = defdat[-index,]

  model_error= glm(default~student+income+balance, data = train, family = binomial)
  pred = ifelse(predict.glm(model_error, newdata = test, type = "response")>0.5, "Yes", "No")
  test_error_5d[i,] = mean(test[,1] != pred)
}
```

	Test Error
Split 1	0.025
Split 2	0.027
Split 3	0.035

Including the *student* dummy variable did not lead to a reduction in test error rate as the validated error did not change significantly from that of Part C.

Question #6

Continuing with the logistic regression model using the Default data we will now compute estimates for the standard errors of *income* and *balance* using bootstrap and the standard glm() formula.

a) Estimated standard errors using glm()

```r
model_6a = glm(default~income+balance, data = defdat, family = binomial)
summary(model_6a)
```

```
Call:
glm(formula = default ~ income + balance, family = binomial,
    data = defdat)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4725  -0.1444  -0.0574  -0.0211   3.7245

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)
```

```
    Null deviance: 2920.6  on 9999  degrees of freedom
Residual deviance: 1579.0  on 9997  degrees of freedom
AIC: 1585

Number of Fisher Scoring iterations: 8
```

Above is a summary of the logistic regression model, again with *default* as the response and *income* and *balance* as predictors. It includes the estimate and standard deviation of each variable's coefficient.

b) Write a function: boot.fn()

```r
boot.fn = function(data, indx){
  model_6b= glm(default~income+balance, data = data, subset = indx, family = binomial)
  coef(model_6b)
}
```

The function above is written to take the inputs *Default* (the data) and *indx* (an index to indicate which subset of observations are to be used), and outputs the coefficient estimates for *income* and *balance*.

c) Using boot() along with boot.fn() to estimate standard errors

```r
defdat = Default
library(boot)
boot(defdat, boot.fn, R = 1000)
```

```

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = defdat, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
         original        bias     std. error
t1* -1.154047e+01 -3.741855e-02 4.298193e-01
t2*  2.080898e-05  1.495111e-07 4.762729e-06
t3*  5.647103e-03  1.743017e-05 2.220594e-04
```

Above is a summary of the estimated bootstrap statistics for each predictor. It includes the estimated coefficient, bias, and standard error. We can see that the estimated standard errors for our coefficients are -1.154e+01 for the intercept, 2.081e-05 for *income*, and 5.647e-03 for *balance*.

d) Compare the estimates of both methods

Comparing the results from our bootstrap model to our logistic model, the estimated coefficients are extremely close while the standard error estimates for the logistic model are slightly larger than those of the bootstrap model.

Question #9

We will now use the Boston housing data set from the MASS library.

a) Estimate the population mean of *medv* and call it $\hat{\mu}$

```r
library(MASS)
boston = Boston
attach(boston)

muhat = mean(medv)
```

The estimate for the population mean of *medv* is 22.5328063.

b) Estimate the standard error of $\hat{\mu}$ and interpret

```r
n = length(medv)
sehat = sd(medv)/sqrt(n)
```

We can interpret this standard error of 0.4088611 as the amount that our mean estimate deviates from the true population mean for the *medv* variable.

c) Estimate the standard error of $\hat{\mu}$ using bootstrap and compare to Part B

```r
bs_mean = matrix(0, 10000, 1)
for(i in 1:10000){
  index = sample(1:n, n, replace = T)
  bootsamp = medv[index]
  bs_mean[i] = mean(bootsamp)
}
se_bs = sd(bs_mean)
```

The bootstrapped standard error of 0.410021 is a good approximation for the standard error found in Part B.

d) Using BS estimate, find 95% CI for mean of *medv* and compare to results from t.test(Boston$medv)

```r
CI = c(muhat-2*sehat, muhat+2*sehat)
CI_t = t.test(medv)$conf.int
```

	95% CI
Parametric	(21.715, 23.351)
t.test	(21.73, 23.336)

Both the parametric confidence interval and the interval computed using t.test are quite similar to one another, with the parametric interval being slightly wider than the t.test interval.

e) Estimate the median value of *medv* and call it $\hat{\mu}_{med}$

```r
mumed = median(medv)
```

The median, or middle value of the data, is 21.2.

f) Estimate the standard error of $\hat{\mu}_{med}$ using bootstrap

```
boot.fn = function(data, indx){
  median(data[indx])
}
boot(medv, boot.fn, R = 1000)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = medv, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
    original   bias    std. error
t1*     21.2 -0.01495   0.3603663
```

We can note that the estimated standard error for the median is slightly lower than the estimated standard error for the mean, implying that the sample median deviates less from the population median than the sample mean deviates from the population mean.

g) Estimate the tenth percentile of *medv* and call it $\hat{\mu}_{0.1}$

```
mu_0.1 = quantile(medv, 0.1)
```

12.75 is the estimate for the tenth percentile of *medv* in Boston suburbs.

h) Use bootstrap to estimate the standard error of $\hat{\mu}_{0.1}$

```
boot.fn = function(data, indx){
  quantile(data[indx], 0.1)
}
boot(medv, boot.fn, R = 1000)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = medv, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
    original  bias    std. error
t1*    12.75  0.0342   0.4905874
```

This final standard error estimate is larger than the standard error estimate for both the mean and median. This implies that the tenth percentile estimate deviates much more from the population tenth percentile, compared to the previous two population parameter estimates.