Lab 06 - Ridge and LASSO Regression

Ken Ye

2023-10-11

1. Ridge Regression and the LASSO

We will use the package `glmnet()` to perform ridge and lasso regression. The main function we will use is `glmnet()`, which can perform ridge, lasso and other types of regression. The syntax for this function is slightly different than the `lm` function. We must pass the `glmnet()` function a matrix for `x` and a vector for `y`, and we won't use the `y~x` syntax as we have been previously.

We will use the `Hitters` data to predict `Salary`. First, we must remove missing values and load the data.

```
## Load the data
library(ISLR)
names(Hitters)
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"
##  [7] "Years"     "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"
## [13] "CWalks"    "League"    "Division"  "PutOuts"   "Assists"   "Errors"
## [19] "Salary"    "NewLeague"
```

```
## Check for NA values
dim(Hitters)
```

```
## [1] 322  20
```

```
sum(is.na(Hitters$Salary))
```

```
## [1] 59
```

```
## Remove NA values
Hitters <- na.omit(Hitters)
dim(Hitters)
```

```
## [1] 263  20
```

```
sum(is.na(Hitters))
```

```
## [1] 0
```

Next, we have to format the data to be compatible with the `glmnet()` function.

```r
x <- model.matrix(Salary~.,Hitters)[,-1]
y <- Hitters$Salary
```

The `model.matrix()` function is very useful for creating `x`, it produces a matrix containing all 19 predictors and also automatically transforms any qualitative variables into dummy variables. This is important, since `glmnet()` can only take numerical, quantitative inputs.

1.1 Ridge Regression

The `alpha` parameter in the `glmnet()` function 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. First, we will fit a ridge regression model.

```r
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

```r
grid <- 10^seq(10, -2, length = 100) # grid of values for lambda param
ridge.mod <- glmnet(x, y, alpha = 0, lambda = grid)
```

By default, `glmnet()` performs ridge regression for an automatically selected range of λ values. Here, we are specifying the grid of values ranging from $\lambda = 10^{10}$ to $\lambda = 10^{-2}$. This range of λ values covers essentially the full range of scenarios we might be interested in, from the null model with only the intercept to the least squares fit. We will see that we can also compute model fits for a particular λ value that is not one of the original grid values.

Note: by default, the `glmnet()` function standardizes the variables so that they are on the same scale. To turn off this setting, use the argument `standardize = FALSE`.

For each value of λ, we have an associated vector of ridge regression coefficients, stored in a matrix that can be accessed by `coef()`. In this case, the coefficients are a 20x100 matrix, with 20 rows (one for each predictor and an intercept) and 100 columns (one for each value of λ).

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

```
## [1]  20 100
```

We expect the coefficient estimates to be much smaller, in terms of l_2 norm, when a large value of λ is used, as compared to when a small value of λ is used. We can look at the results for a specific value of λ, $\lambda = 11,498$.

```r
ridge.mod$lambda[50]
```

```
## [1] 11497.57
```

```r
coef(ridge.mod)[,50]
```

```
##   (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)) # intercept not regularized
```

```
## [1] 6.360612
```

In contrast, here are the coefficients when $\lambda = 705$ and their l_2 norm. There is a much larger l_2 norm here, since we have a smaller value of λ.

```
ridge.mod$lambda[60]
```

```
## [1] 705.4802
```

```
coef(ridge.mod)[,60]
```

```
##  (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))
```

```
## [1] 57.11001
```

We can use the `predict()` function for a number of different tasks. For example, we can obtain ridge regression coefficients for a new value of λ, say $\lambda = 50$.

```
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
```

Now we can split the samples into a training set and a test set so that we can estimate the test error of ridge regression and the lasso. There are multiple ways that we have looked at for splitting the data into training and test sets. Make sure to set the seed so that we can reproduce the results.

```
set.seed(17)
train <- sample(1:nrow(x), nrow(x)/2)
test <- (-train)
y.test <- y[test]
```

Next, we fit a ridge regression model on the training set and evaluate its MSE on the test set, using $\lambda = 4$. Note that we use the `predict()` function again. This time, we are getting 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) ## calculate MSE
```

```
## [1] 114838.3
```

The test MSE is very large. Note that if we had just fit a model with only an intercept, we would have predicted each test observation using the mean of the training observations. Then, our test MSE would be:

```
mean((mean(y[train]) - y.test)^2)
```

```
## [1] 194103.2
```

We could also get the same result by fitting a ridge regression model with a very large value of λ. For example:

```
ridge.pred <- predict(ridge.mod, s=1e10, newx = x[test,])
mean((ridge.pred - y.test)^2)
```

```
## [1] 194103.1
```

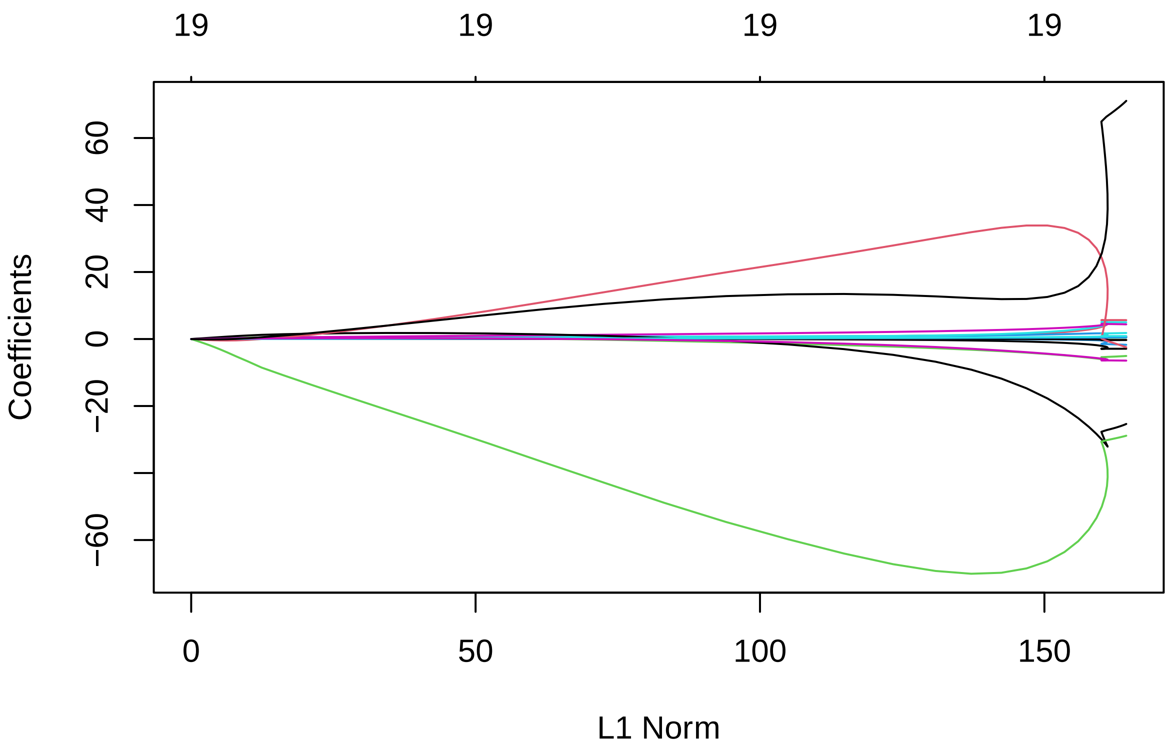
We can conclude that fitting a ridge regression model with $\lambda = 4$ leads to a much lower test MSE than fitting the model with just an intercept. We can now check whether there is any benefit to performing ridge regression with $\lambda = 4$ instead of just performing least squares regression.

Recall that least squares regression is just ridge regression with $\lambda = 0$.

```
ridge.pred <- predict(ridge.mod, s=0, x = x[train,], y = y[train],
                      newx = x[test,], exact = T)
mean((ridge.pred - y.test)^2)
```

```
## [1] 122836.2
```

```
plot(ridge.mod)
```



```
lm(y~x, subset = train)
```

```
##
## Call:
## lm(formula = y ~ x, subset = train)
##
## Coefficients:
## (Intercept)       xAtBat        xHits       xHmRun        xRuns         xRBI
##    356.5978      -2.8971       5.6352      -5.0483       0.5984       4.9963
##      xWalks       xYears      xCAtBat       xCHits      xCHmRun       xCRuns
##      4.3838     -25.2886      -0.2584       0.3956      -1.7330       1.8087
##       xCRBI      xCWalks     xLeagueN   xDivisionW     xPutOuts     xAssists
##      0.7759      -0.3048      -2.5059     -28.8259       0.4657       0.8776
##     xErrors  xNewLeagueN
##     -6.4521      71.2296
```

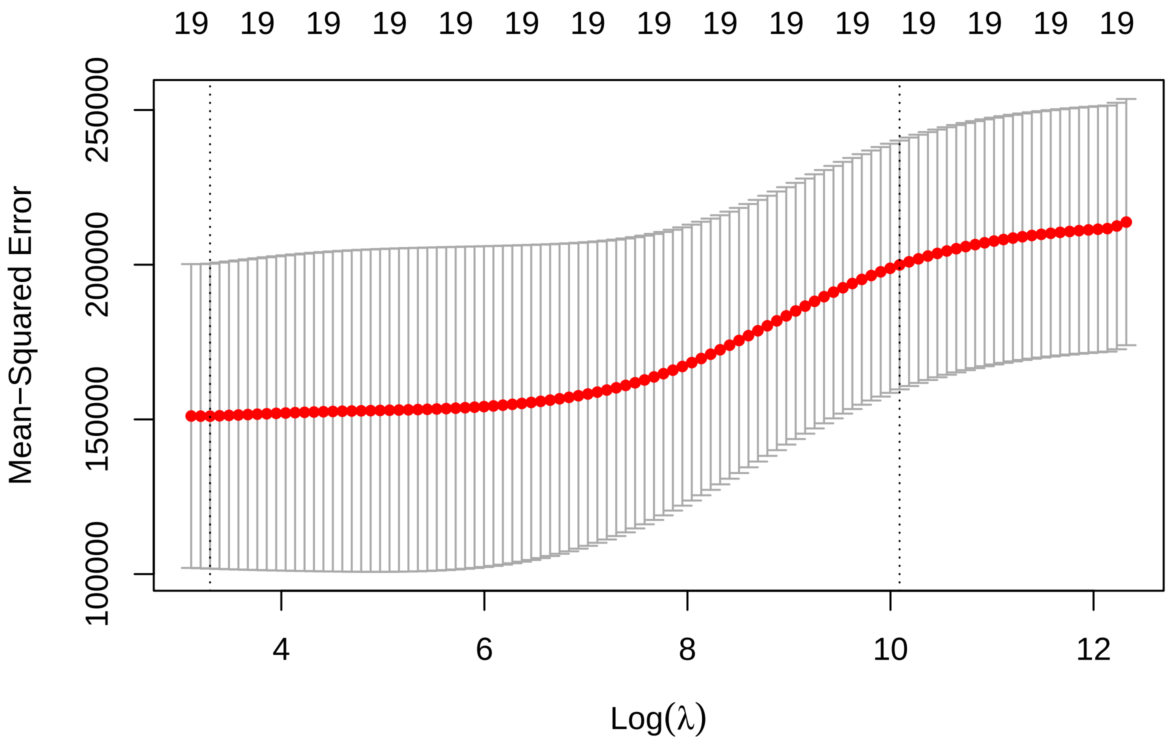
```
predict(ridge.mod, s=0, exact = T, x = x[train,], y = y[train],
        type = 'coefficients')[1:20,]
```

```
## (Intercept)       AtBat        Hits       HmRun        Runs         RBI
## 356.5933262  -2.8970003   5.6346702  -5.0488384   0.5986376   4.9965087
##       Walks       Years      CAtBat       CHits      CHmRun       CRuns
##   4.3836138 -25.2869225  -0.2584242   0.3958402  -1.7326146   1.8085757
##        CRBI      CWalks     LeagueN   DivisionW     PutOuts     Assists
##   0.7756573  -0.3047745  -2.5017231 -28.8257346   0.4657154   0.8776513
##      Errors  NewLeagueN
##  -6.4523921  71.2268300
```

In general, if we want to fit a least squares model with no penalty, then we should use the `lm()` function, since that function provides more useful outputs, such as standard errors and p-values for the coefficients.

A more principled method to choose the value of λ that we should use in general is to use cross validation. We can do this by using the built-in cross-validation function, `cv.glmnet()`. By default, the function performs 10-fold cross validation, but we can change this using the argument `folds`. Note that we need to set a random seed first so that our results are reproducible, since the choice of the cross-validation folds is random.

```
set.seed(1)
cv.out <- cv.glmnet(x[train,], y[train], alpha = 0)
plot(cv.out)
```



```
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 27.0624
```

In the plot above, the numbers across the top of the plot are the number of nonzero coefficient estimates for the model. Ridge regression does not set coefficients to 0, so all variables are included in every model.

From the cross validation, we can 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)
```

```
## [1] 106331.5
```

This represents a further improvement over the test MSE that we got using $\lambda = 4$. Finally, we can refit our ridge regression model on the full data set, using the value of λ chosen by cross-validation and we can examine the coefficient estimates.

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

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  7.824917e+01 -6.507281e-01  2.692086e+00 -1.379582e+00  1.033875e+00
##           RBI         Walks         Years        CAtBat         CHits
##  7.244559e-01  3.316575e+00 -8.855335e+00 -3.826829e-04  1.335107e-01
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##  6.927946e-01  2.879479e-01  2.536312e-01 -2.672665e-01  5.267305e+01
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -1.225771e+02  2.629501e-01  1.655801e-01 -3.659976e+00 -1.743993e+01
```

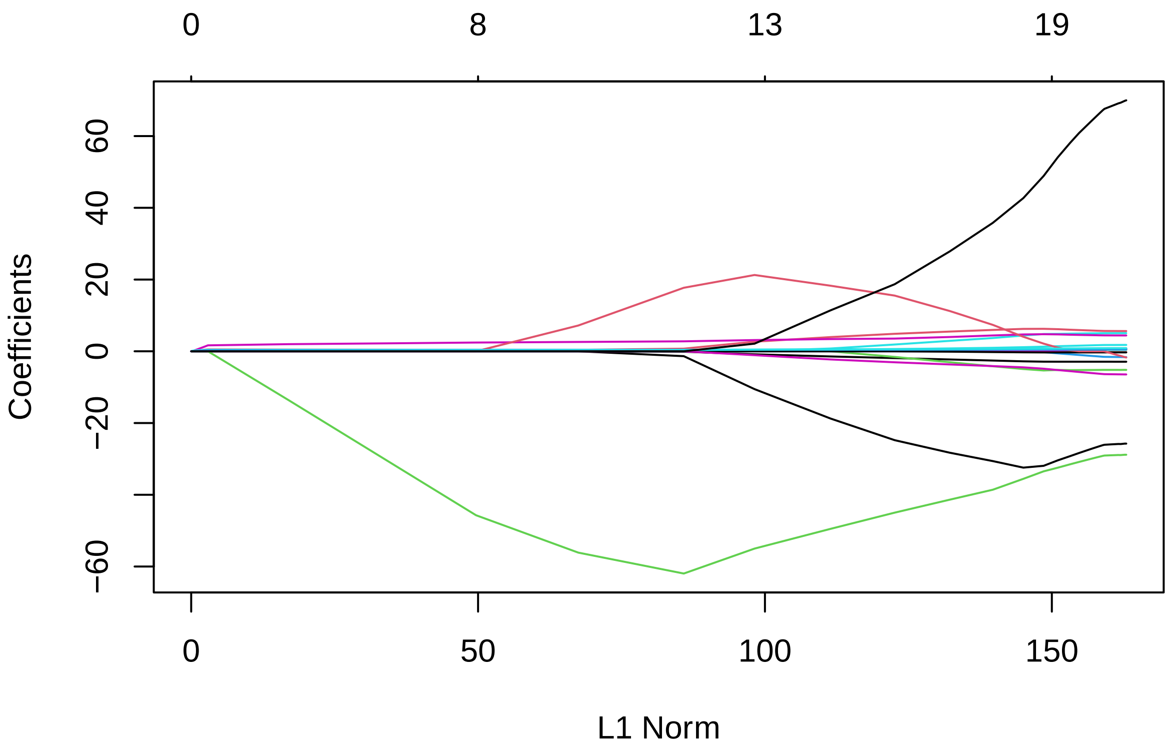
As expected, none of the coefficients are 0 - ridge regression does not perform variable selection!

1.2 The Lasso

We saw that ridge regression with a good choice of λ can outperform least squares as well as the null model on the `Hitters` data set. We can now explore whether the lasso can yield either a more accurate or a more interpretable model than ridge regression. To fit a lasso model, we can again use `glmnet()`. This time, we will use the argument `alpha = 1`. Other than the change in the α parameter, we can proceed as before:

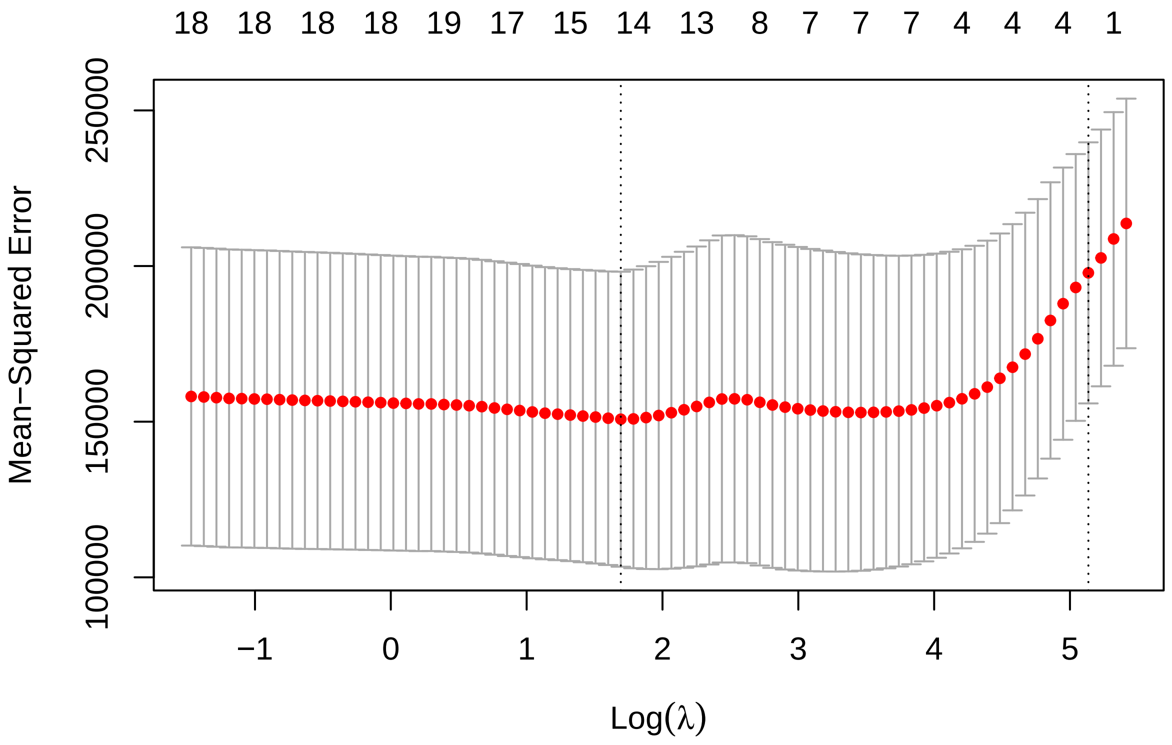
```r
lasso.mod <- glmnet(x[train,], y[train], alpha = 1, lambda = grid)
plot(lasso.mod)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```



We can see from the coefficient plot that depending on the choice of the tuning parameter, some of the coefficients will be exactly equal to 0. We can now compute cross-validation and compute the associated test error.

```r
set.seed(1)
cv.out <- cv.glmnet(x[train,], y[train], alpha = 1)
plot(cv.out)
```



```r
bestlam <- cv.out$lambda.min
lasso.pred <- predict(lasso.mod, s = bestlam, newx = x[test,])
mean((lasso.pred - y.test)^2)
```

```
## [1] 108090.7
```

This is much lower than the test set MSE that we got with the null model and of least squares. It is very similar to the test MSE of ridge regression with the λ chosen by cross validation.

However, the lasso results in sparse coefficient estimates, which can be a substantial advantage over ridge regression. Here, we see that 12 of the 19 coefficients are exactly 0. So the lasso model with λ chosen by cross validation results in a model with only 7 variables.

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

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##   57.98365633   -0.78631371    3.98853475    0.00000000    0.00000000
##           RBI         Walks         Years        CAtBat         CHits
##    0.00000000    3.35011623   -5.19635607    0.00000000    0.00000000
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##    0.21070493    0.40649806    0.41599404   -0.21350141   27.49011144
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -118.63996612    0.25204510    0.03295392   -0.87259198    0.00000000
```

Exercise

In this exercise, we will predict the number of applications received using the other variables in the College data set.

(a) Split the data set into a training set and a test set.

```
names(College)
```

```
##  [1] "Private"     "Apps"        "Accept"      "Enroll"      "Top10perc"  
##  [6] "Top25perc"   "F.Undergrad" "P.Undergrad" "Outstate"    "Room.Board" 
## [11] "Books"       "Personal"    "PhD"         "Terminal"    "S.F.Ratio"  
## [16] "perc.alumni" "Expend"      "Grad.Rate"
```

```
dim(College)
```

```
## [1] 777  18
```

```
# remove null obs
College <- na.omit(College)
dim(College)
```

```
## [1] 777  18
```

```
sum(is.na(College))
```

```
## [1] 0
```

```
x <- model.matrix(Apps ~ .,College)[,-1]
y <- College$Apps
```

```
# split data
set.seed(1)
train <- sample(1:nrow(x), nrow(x)/2)
test <- setdiff(1:nrow(x), train)
y.test <- y[test]
```

(b) Fit a linear model using least squares on the training set, and report the test error obtained.

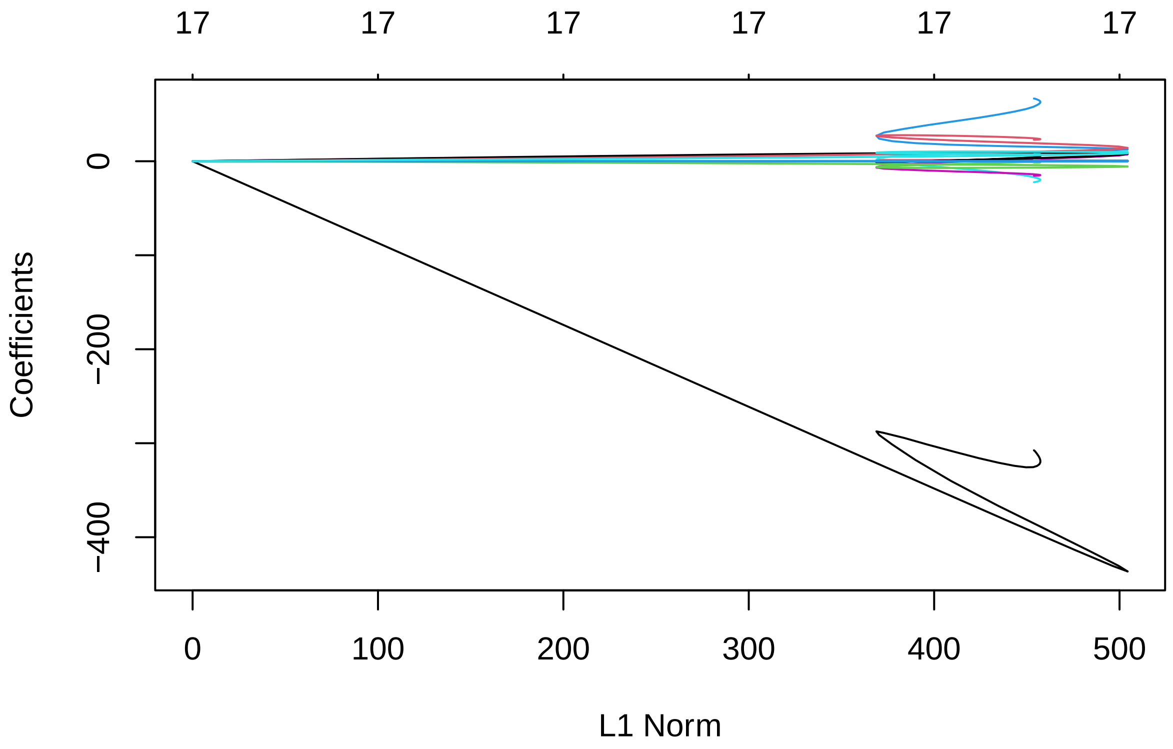
```
# linear regression
lm <- lm(y ~ x, subset = train)
lm.pred <- predict(lm, newx = x[test,])
```

```r
# calculate MSE
mean((lm.pred - y.test[1:length(lm.pred)])^2)
```

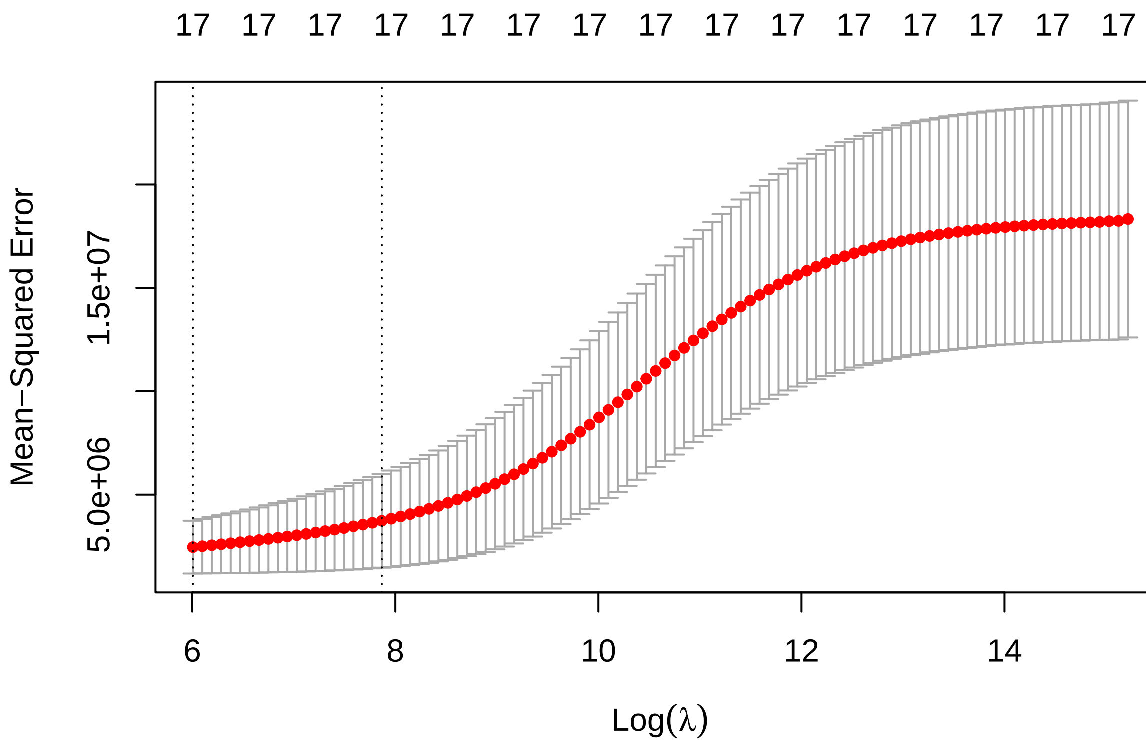
```
## [1] 28865826
```

(c) Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained.

```r
ridge.mod <- glmnet(x[train,], y[train], alpha = 0, lambda = grid)
plot(ridge.mod)
```



```r
# cv
set.seed(1)
cv.out <- cv.glmnet(x[train,], y[train], alpha = 0)
plot(cv.out)
```



```r
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 405.8404
```

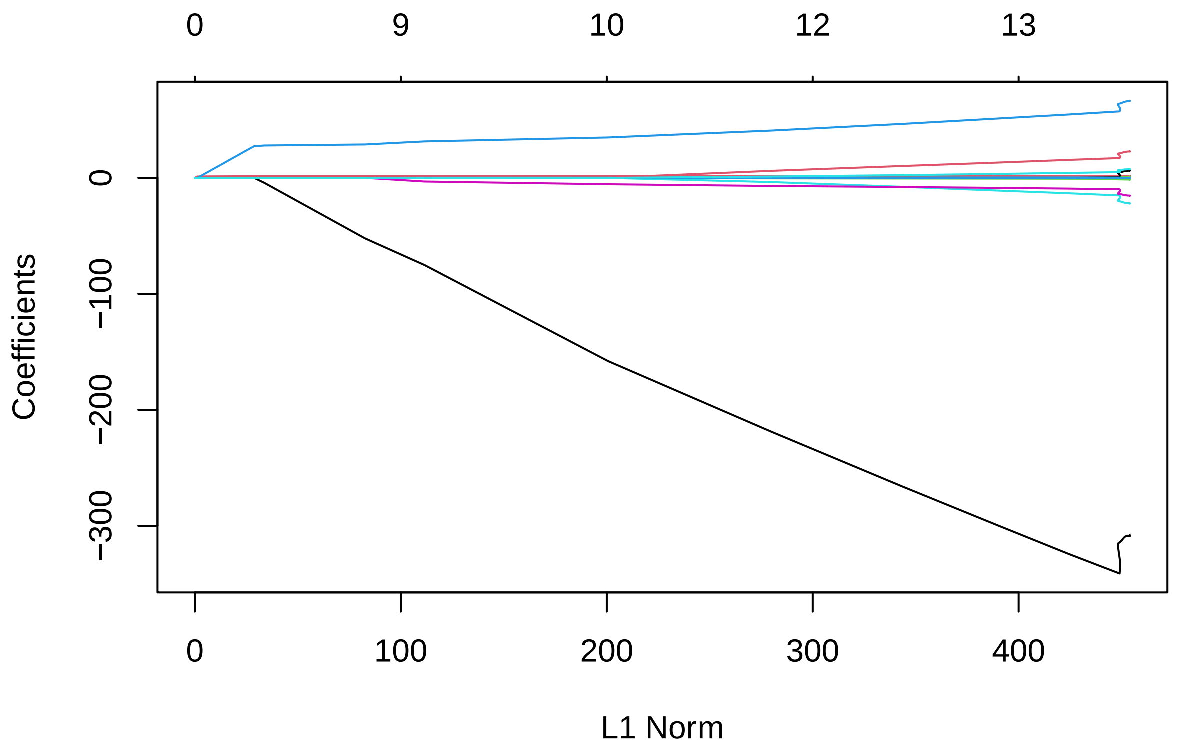
```r
ridge.pred <- predict(ridge.mod, s = bestlam, newx = x[test,])
mean((ridge.pred - y.test)^2)
```

```
## [1] 976268.9
```

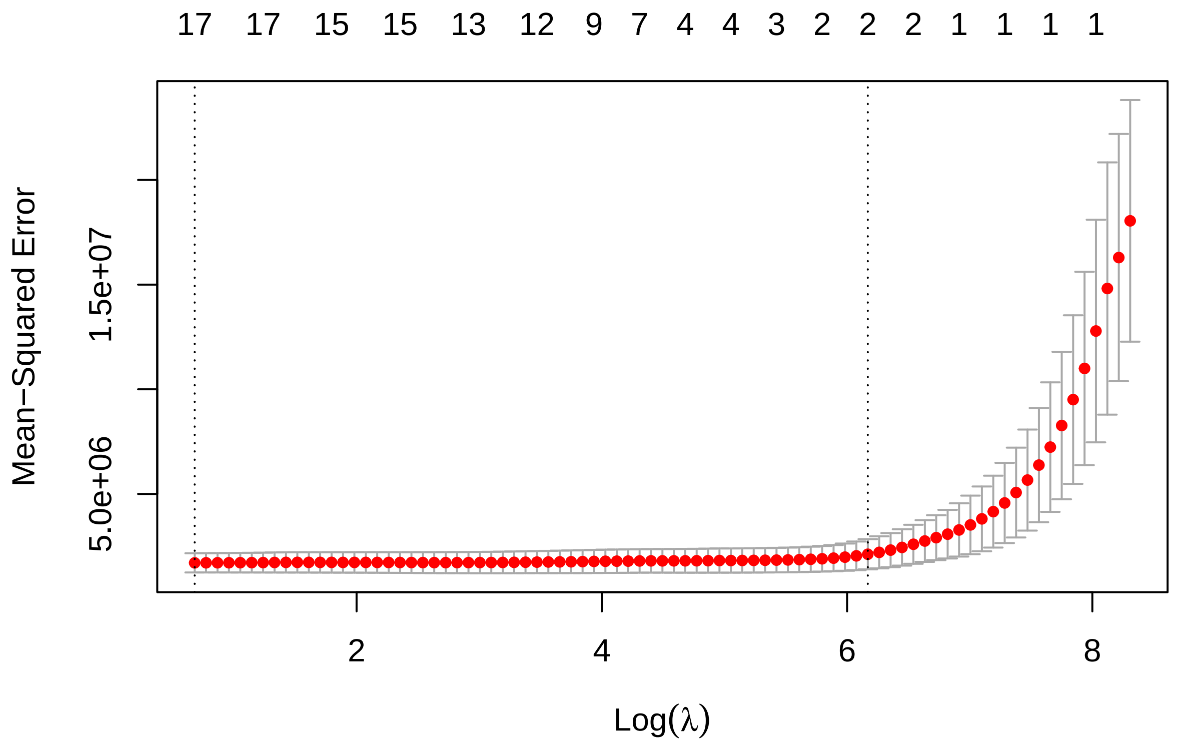
(d) Fit a lasso model on the training set, with λ chosen by cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

```r
lasso.mod <- glmnet(x[train,], y[train], alpha = 1, lambda = grid)
plot(lasso.mod)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```



```
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)
```

```
## [1] 1116252
```

```
out <- glmnet(x, y, alpha = 1, lambda = grid)
lasso.coef <- predict(out, type = "coefficients", s = bestlam)
lasso.coef
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) -468.96036214
## PrivateYes  -491.47351867
## Accept         1.57117027
## Enroll        -0.76858284
## Top10perc     48.25732969
## Top25perc    -12.94716203
## F.Undergrad    0.04293538
## P.Undergrad    0.04406960
## Outstate      -0.08340724
## Room.Board     0.14963683
## Books          0.01549548
## Personal       0.02915128
## PhD           -8.42538012
## Terminal      -3.26671319
## S.F.Ratio     14.61167079
## perc.alumni   -0.02612797
## Expend         0.07718333
## Grad.Rate      8.31579869
```

The number of the non-zero coefficient estimates is 17.

(e) Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these three approaches?

Our prediction of the number of college applications received is not great, as indicated by the high test MSE's. Among the three, least squares had the highest MSE, 28865826, followed by Lasso's, 1116252. Ridge achieve the lowest MSE among the three, of 976268.9.