Homework 3

Author: Tanay Mukherjee
Date: 8th May 2020

1. In this problem, you will consider choosing the tuning parameters for both ridge regression and the lasso, using 10-fold cross-validation. First download the les \main.R", \plotfuns.R", and \bstar.Rdata". The rst line of the le \main.R" has you install the package glmnet. Once you have done this (i.e., once you have installed this package), you can comment this line out.

   We begin with a true signal bstar. Although this is stored as a vector of length p = 2500, bstar really represents an image of dimension 50 50. You can plot it by calling plot.image(bstar) .

   This image is truly sparse, in the sense that 2084 of its pixels have a value of 0, while 416 pixels have a value of 1. You can think of this image as a toy version of an MRI image that we are interested in collecting.

   Suppose that, because of the nature of the machine that collects the MRI image, it takes a long time to measure each pixel value individually, but it's faster to measure a linear combination of pixel values. We measure n = 1300 linear combinations, with the weights in the linear combination being random, in fact, independently distributed as N(0; 1). These measurements are given by the entries of the vector

   X % % b s t a r

   in our R code. Because the machine is not perfect, we don't get to observe this directly, but we see a noisy version of this. Hence, in terms of our R code, we observe

   y = X % % b s t a r + rnorm(n , sd =5).

   Now the question is: can we model y as a linear combination of the columns of X to recover some coefficient vector that is close to bstar? Roughly speaking, the answer is yes. Key points here: although the number of measurements n = 1300 is smaller than the dimension p = 2500, the true vector bstar is sparse, and the weights in a linear combination are i.i.d normal. This is the idea behind the eld of compressed sensing. Below, you can nd several clips regarding the history, motivation, and applications of compressed sensing:

   > Robust Compressed Sensing: How Undersampling Introduces Noise and What We Can Do About It (minutes 2-16). https://www.youtube.com/watch?v=ThiAk_ n-8HI
   >
   > Compressed Sensing: Recovery, Algorithms, and Analysis (rst 4 minutes). https: //www.youtube.com/watch?v=mgCIKnMgBmk
   >
   > Compressive Sensing (minutes 5-16). https://www.youtube.com/watch?v=RvMgVv-xZhQ

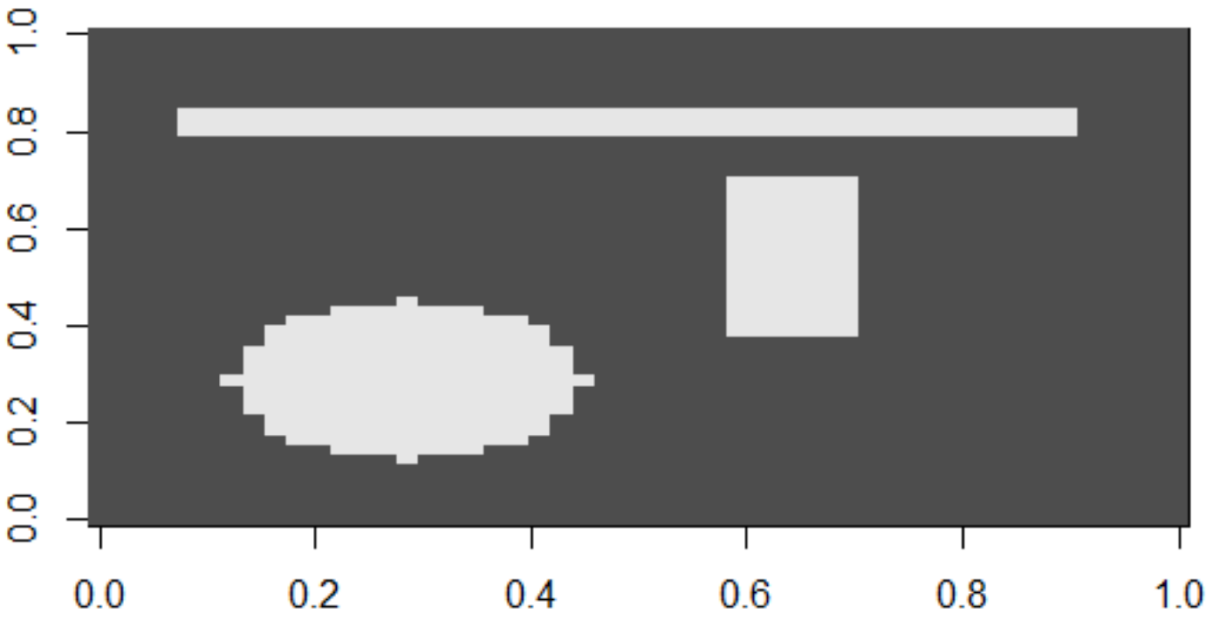
   The le \main.R" is setup to perform ridge regression of y on X, and the lasso of y on X, with the tuning parameter for each method selected by cross-validation. You will fill in the missing pieces. It's helpful to read through the whole le to get a sense of what's to be accomplished. Try to understand all the parts, even if it doesn't seem related to what you have to fill in; this should be good practice for working with R in the future, etc.

Let's first load the necessary libraries and btsart.Rdata. Next, let's see the image we are going to work on.

Code:

```
> library(glmnet)
> source("plotfuns.R")
> load("bstar.Rdata")
> plot.image(bstar)
```

Output:



a. Fill in the missing parts. There are 2 missing parts marked by # TODO. When you're getting started, it might be helpful to read the documentation for the glmnet function, which you will use to perform ridge regression and the lasso.

Code:

```
> p = length(bstar)
> set.seed(0)
> n = 1300
> X = matrix(rnorm(n*p),nrow=n)
> y = X%*%bstar + rnorm(n,sd=5)
> K = 10
> d = ceiling(n/K)
> set.seed(0)
> i.mix = sample(1:n)

> # Tuning parameter values for lasso, and ridge regression
> lam.las = c(seq(1e-3,0.1,length=100),seq(0.12,2.5,length=100))
> lam.rid = lam.las*1000
> nlam = length(lam.las)

> # These two matrices store the prediction errors for each
> # observation (along the rows), when we fit the model using
> # each value of the tuning parameter (along the columns)

> e.rid = matrix(0,n,nlam)
> e.las = matrix(0,n,nlam)
```

```
> for (k in 1:K) {
+   cat("Fold",k,"\n")
+
+   folds=(1+(k-1)*d):(k*d);
+   i.tr=i.mix[-folds]
+   i.val=i.mix[folds]
+
+   X.tr = X[i.tr,]   # training predictors
+   y.tr = y[i.tr]    # training responses
+   X.val = X[i.val,] # validation predictors
+   y.val = y[i.val]  # validation responses
+
+   # TODO1
+   # Now use the function glmnet on the training data to get the
+   # ridge regression solutions at all tuning parameter values in
+   # lam.rid, and the lasso solutions at all tuning parameter
+   # values in lam.las
+
+   a.rid = glmnet(X.tr, y.tr, alpha = 0, lambda = lam.rid) # for the
ridge regression solutions, use alpha=0
+   a.las = glmnet(X.tr, y.tr, alpha = 1, lambda = lam.las) # for the
lasso solutions, use alpha=1
+
+
+   # Here we're actually going to reverse the column order of the
+   # a.rid$beta and a.las$beta matrices, because we want their columns
+   # to correspond to increasing lambda values (glmnet's default makes
+   # it so that these are actually in decreasing lambda order), i.e.,
+   # in the same order as our lam.rid and lam.las vectors
+   rid.beta = as.matrix(a.rid$beta[,nlam:1])
+   las.beta = as.matrix(a.las$beta[,nlam:1])
+
+   yhat.rid = X.val%*%rid.beta
+   yhat.las = X.val%*%las.beta
+
+   e.rid[i.val,] = (yhat.rid-y.val)^2
+   e.las[i.val,] = (yhat.las-y.val)^2
+ }
```

Output:

```
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Fold 6
Fold 7
Fold 8
Fold 9
Fold 10
```

```
# TODO2
```

```
> cv.rid = apply(e.rid, 2, mean)
> cv.las = apply(e.las, 2, mean)
> se.rid = apply(e.rid, 2,function (x) sd(x)/(n^0.5))
> se.las = apply(e.las, 2,function (x) sd(x)/(n^0.5))
```

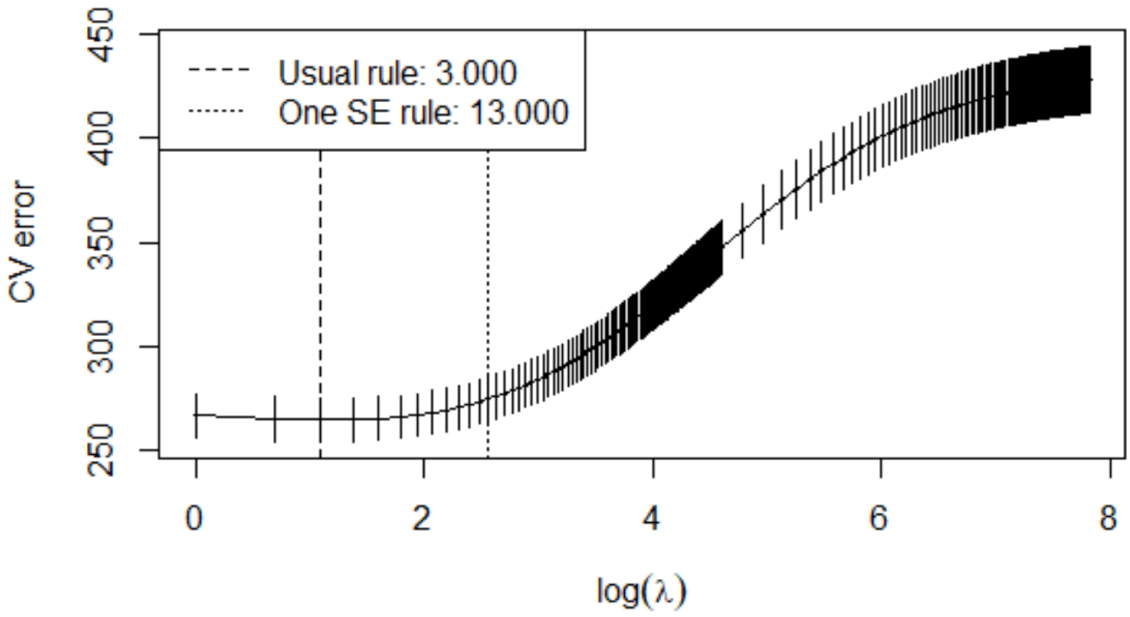
b. Plot the cross-validation error curves for each of ridge regression and the lasso. You can do this using the function plot.cv, as demonstrated by the code at the end. For both ridge regression and the lasso, what value of λ is chosen by the usual rule? What value is chosen by the one standard error rule? Which method, ridge regression or the lasso, has a smaller minimum cross-validation error?

Code:

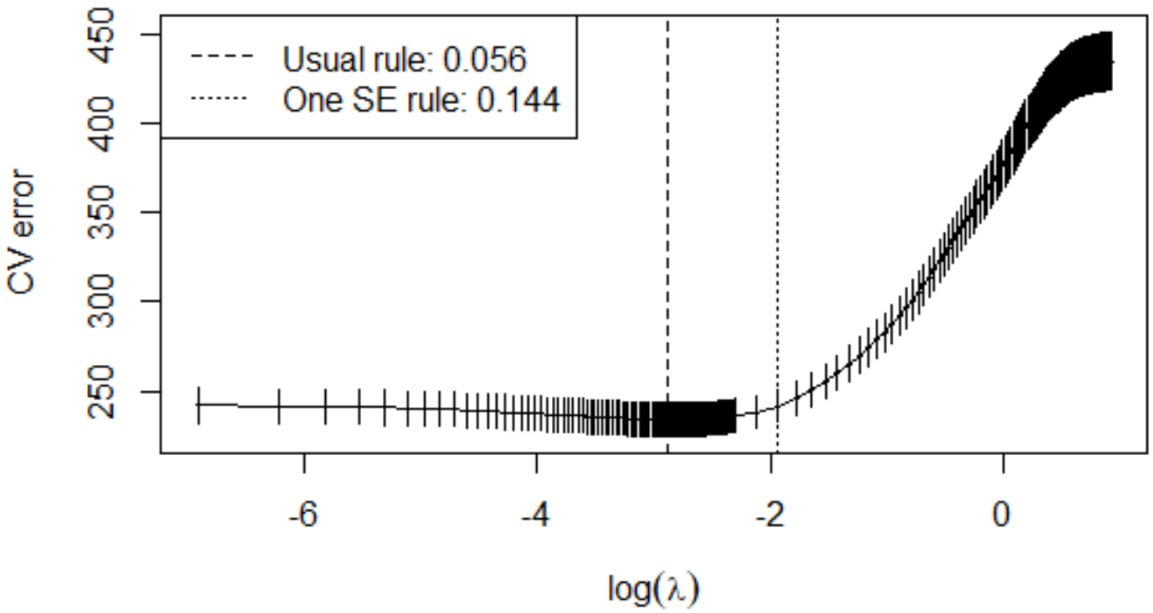
```
> # Usual rule for choosing lambda
> i1.rid = which.min(cv.rid)
> i1.las = which.min(cv.las)

> # one-standard-error rule for choosing lambda
> i2.rid = max(which(cv.rid <= cv.rid[i1.rid]+se.rid[i1.rid]))
> i2.las = max(which(cv.las <= cv.las[i1.las]+se.las[i1.las]))

> #For Ridge
> plot.cv(cv.rid,se.rid,lam.rid,i1.rid,i2.rid)

> #For Lasso
> plot.cv(cv.las,se.las,lam.las,i1.las,i2.las)
```

Output:

(a) Ridge



(b) Lasso



Result:

Values of Lambda (λ)		
	Ridge	**Lasso**
Usual Rule	3	0.056
One Standard Rule	13	0.144

```
> #Minimum cross-validation error
> #For Ridge
> min(cv.rid)
[1] 264.6036

> #For Lasso
> min(cv.las)
[1] 233.9604
```

Therefore, lasso regression has the smaller minimum cross-validation error.

c. Now run ridge regression and the lasso on the entire data set X,y, for the same tuning parameter values as you did before. Save the objects returned by glmnet as a.rid, a.las, respectively. Plot the coefficient images corresponding to the values of chosen by 10-fold CV, for each of ridge regression and the lasso. For this, you'll want to use the indices that you computed in parts (a) and (b), i1.rid, i1.las (usual rule), as well as the coefficients a.rid$beta, a.las$beta that you just computed. Which image looks better? What is the difference between the ridge regression images and the lasso images? Which do you think matches the true image bstar more closely?

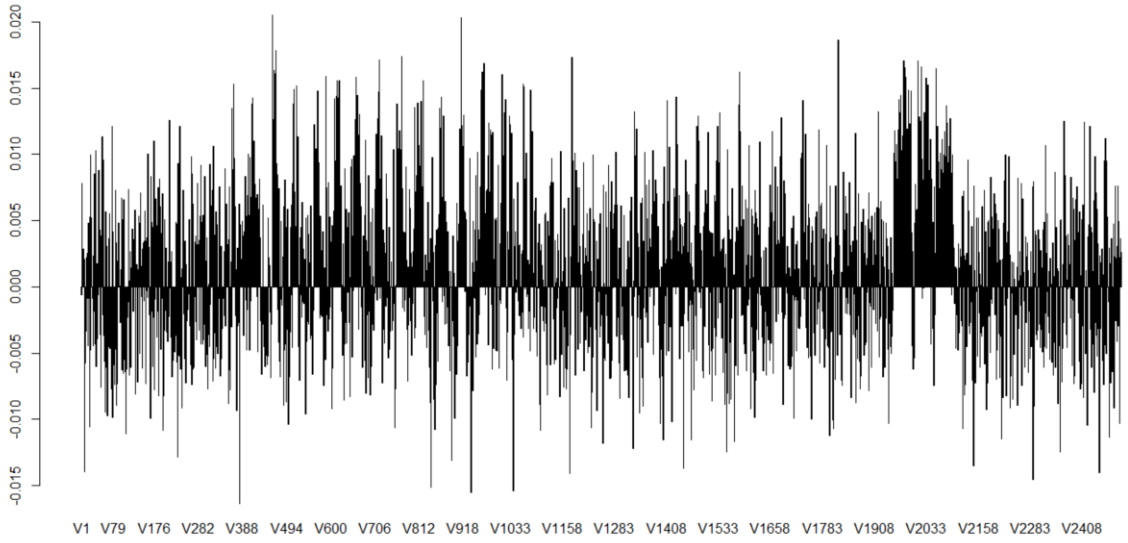
1. Checking coefficients:

Code:

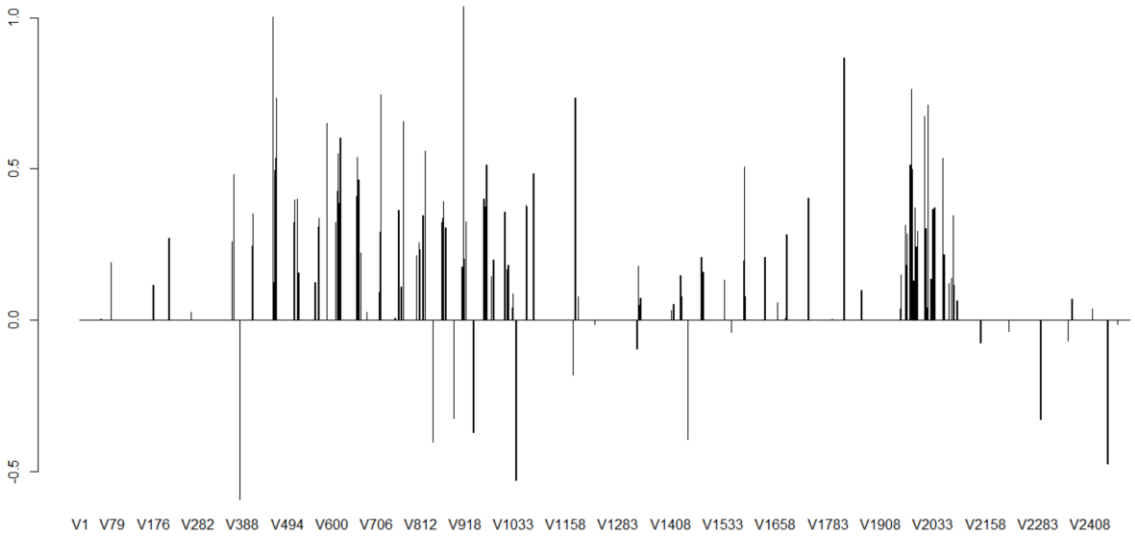
```
> a.rid = glmnet(X, y, alpha = 0, lambda = lam.rid)
> a.las = glmnet(X, y, alpha = 1, lambda = lam.las)

> #For Ridge
> barplot(height = a.rid$beta[,i1.rid])

> #For Lasso
> barplot(height = a.las$beta[,i1.las])
```

Output:

(a) Ridge



(b)Lasso



Explanation:
Lasso looks much better as it is much sparser. The difference is that Ridge regression only shrinks the coefficients toward zero but not zero, while lasso will shrink non-significant coefficients to zero. Since, we know that the true bstar image is very sparse, coefficients given by lasso matches the true bstar better.

2. Plotting the image

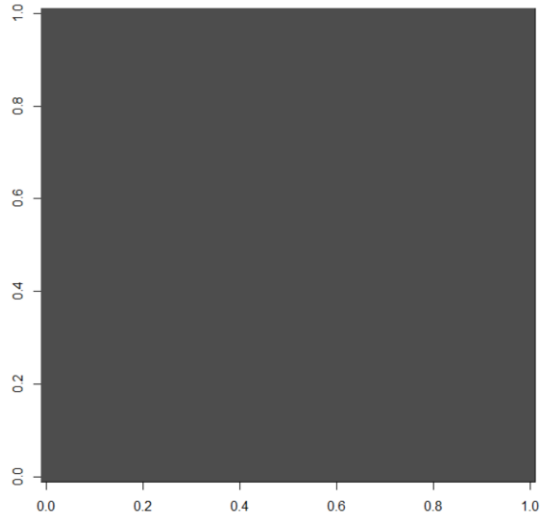
(a) Using all the lambda values

Code:

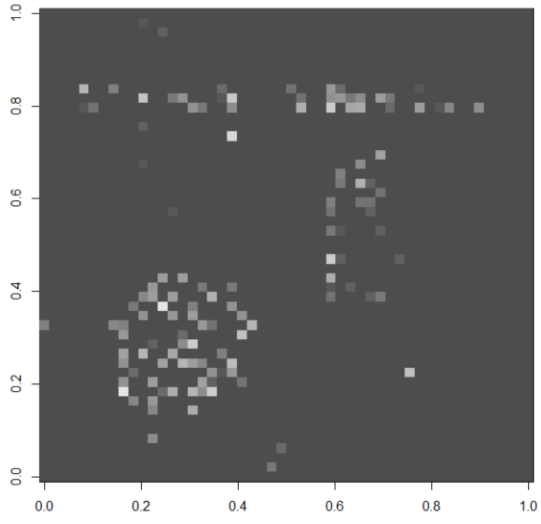
```
> plot.image(a.rid$beta[, i1.rid])
> plot.image(a.las$beta[, i1.las])
```

Output:

Ridge



Lasso

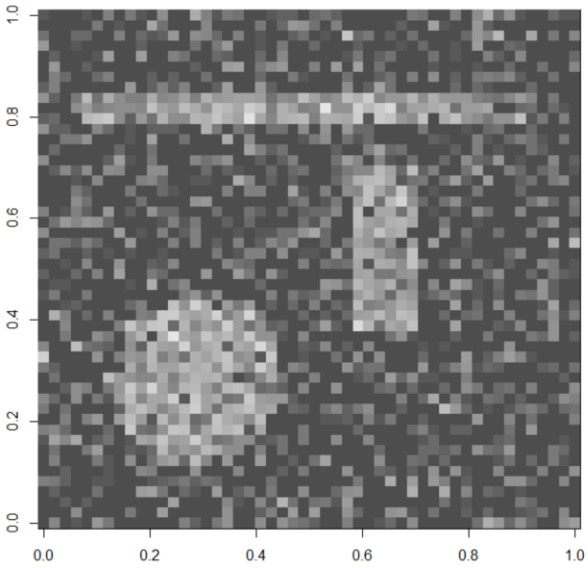


(b) Using the best lambda values

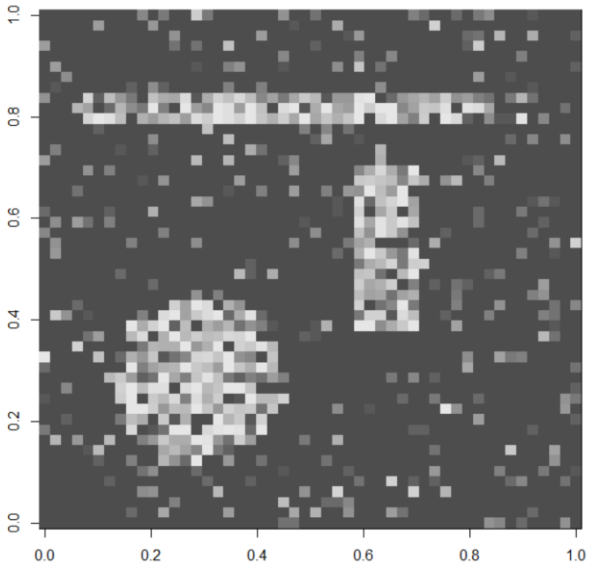
Code:

```
> # Use the lambda values from the usual rule we calculated in step b
> a.rid = glmnet(X.tr, y.tr, alpha = 0, lambda = 3)
> a.las = glmnet(X.tr, y.tr, alpha = 1, lambda = 0.056)


> plot.image(a.rid$beta)
> plot.image(a.las$beta)
```

Ridge



Lasso



Explanation:

From the bar plots of coefficients, we expected the lasso to show better results. To verify this, let us plot the bstar image again and observe the result. It is pretty clear that while Ridge gives no result matching the original image, Lasso still is able to pick the result close to our original image as it overcomes the disadvantage of Ridge regression by not only punishing high values of the coefficients β but actually setting them to zero if they are not relevant.

Now, for next step I decide to consider the best lambda from the usual rule we have computed earlier. Since, we are not into the cross validation anymore I decided to apply the same on all lambda, and we see how ridge appears. We can see the bstar image there in the backdrop with too much noise, i.e., features which are not significant still influencing the outcome whereas with lasso, it is getting clearer and is close to the original image with little distraction

We must think why it is easy to remove coefficients in lasso than ridge. It is because in the ridge, the coefficients of the linear transformation are normally distributed and, in the lasso, they are Laplace distributed. In the lasso, this makes it easier for the coefficients to be zero and therefore easier to eliminate some of your input variable as not contributing to the output.

d. Look at the squared error between the ridge regression and the lasso coefficients that you computed in (c) and the true coefficient vector bstar. What has the lowest squared error?

Code:

```
> #Squared error
> #For Ridge
> sum((bstar - a.rid$beta[,i1.rid])^2)
[1] 409.3363

> For Lasso
> sum((bstar - a.las$beta[,i1.las])^2)
[1] 364.2909
```

Conclusion:
We can clearly see that lasso has the lowest squared error: 364.2909.

e. Look at the absolute error between the ridge regression and the lasso coefficients that you computed in (c) and the true coefficient vector bstar. What has the lowest absolute error?

Code:

```
> #Absolute error
> #For Ridge
> sum(abs(bstar - a.rid$beta[,i1.rid]))
[1] 420.6706

> #For Lasso
> sum(abs(bstar - a.las$beta[,i1.las]))
[1] 387.7658
```

Conclusion:
We can clearly see that lasso has the lowest absolute error: 387.7658.