# Stats191 Homework 5

## Question 1.

Fot this problem, use the HIV resistance data in the penalized regression slides.

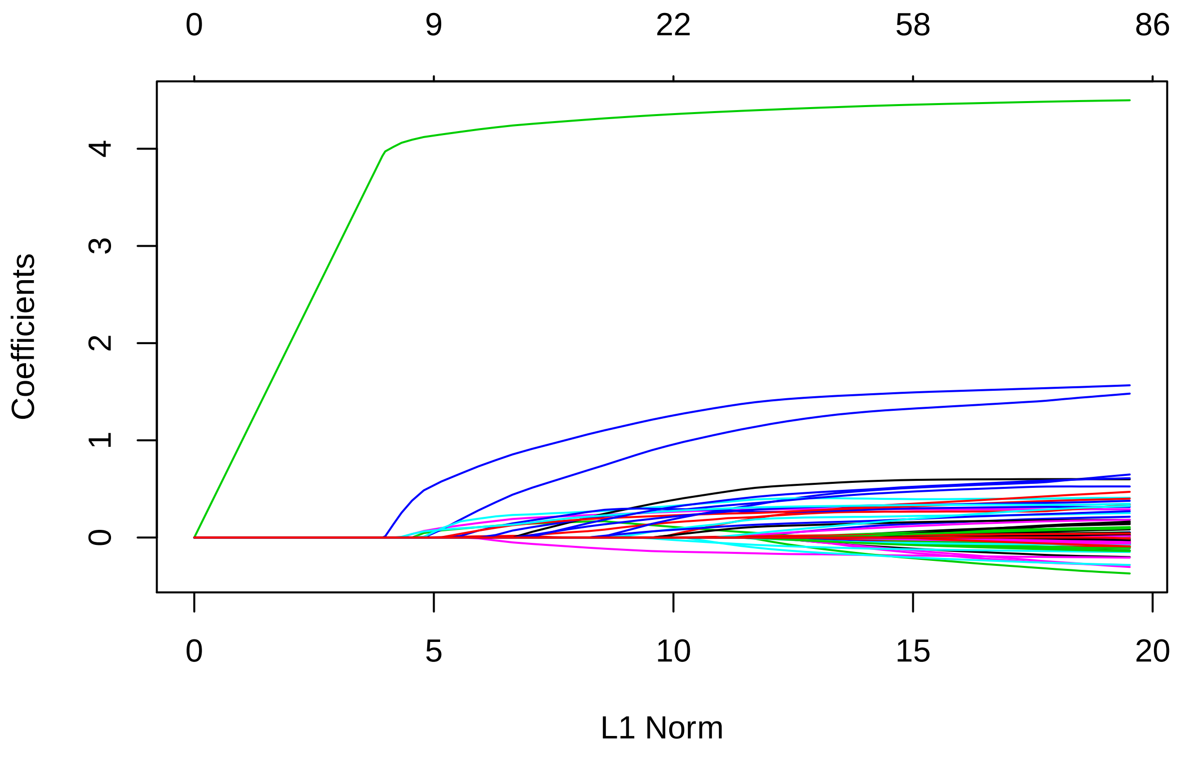
```
X_HIV = read.table('http://stats191.stanford.edu/data/NRTI_X.csv', header=FALSE, sep=',')
Y_HIV = read.table('http://stats191.stanford.edu/data/NRTI_Y.txt', header=FALSE, sep=',')
set.seed(0)
Y_HIV = as.matrix(Y_HIV)[,1]
X_HIV = as.matrix(X_HIV)
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.0
```

```
G = glmnet(X_HIV, Y_HIV)
plot(G)
```




```
nrow(X_HIV)
```

```
## [1] 633
```

```
length(Y_HIV)
```

```
## [1] 633
```

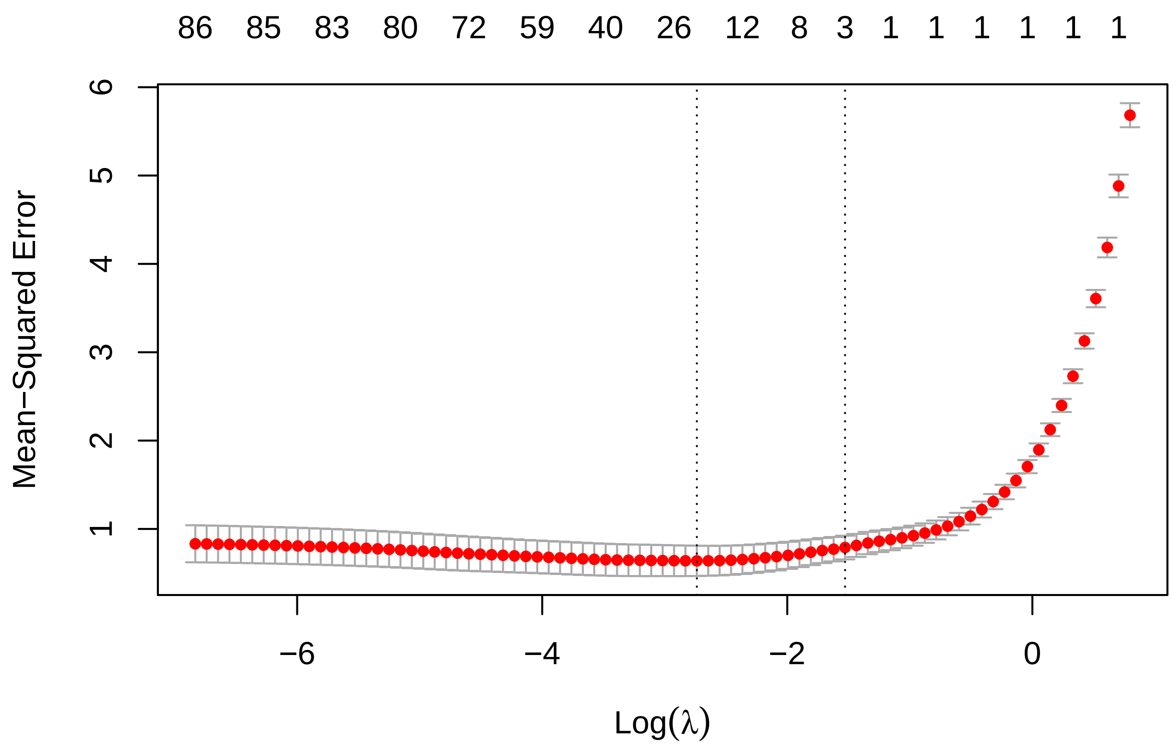
1. Randomly split the data in half.

```
sample <- sample.int(n = nrow(X_HIV), size = floor(0.5*nrow(X_HIV)), replace = F)
train_X <- X_HIV[sample, ]
test_X <- X_HIV[-sample, ]

train_Y <- Y_HIV[sample]
test_Y <- Y_HIV[-sample]
```

2. Using the first half of the data, fit the LASSO with cross-validation using cv.glmnet. Extract the coefficients at lambda.min and lambda.1se. Are the estimates sparse or are all coefficients non-zero? (Answer will depend somewhat on the seed you use – set an integer seed and save it.)

```
CV = cv.glmnet(train_X, train_Y)
plot(CV)
```




```
CV$lambda.1se
```

```
## [1] 0.2168411
```

```
CV$lambda.min
```

```
## [1] 0.06469774
```

The estimates for lambda.1se are sparse and most of the coefficients are zero.

```
beta.hat.1se = coef(G, s=CV$lambda.1se)
beta.hat.1se
```

```
## 92 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) 1.016399e+00
## V1          .
## V2          .
## V3          .
```

```
## V4          .
## V5          .
## V6          .
## V7          .
## V8          .
## V9          .
## V10         .
## V11         .
## V12         .
## V13         .
## V14         .
## V15         .
## V16         .
## V17         1.401608e-01
## V18         .
## V19         4.069764e-05
## V20         .
## V21         .
## V22         .
## V23         .
## V24         .
## V25         .
## V26         .
## V27         .
## V28         .
## V29         .
## V30         .
## V31         .
## V32         .
## V33         .
## V34         .
## V35         .
## V36         .
## V37         .
## V38         .
## V39         .
## V40         .
## V41         .
## V42         .
## V43         .
## V44         .
## V45         .
## V46         .
## V47         .
## V48         .
## V49         .
## V50         .
## V51         .
## V52         .
## V53         .
## V54         .
## V55         .
## V56         .
## V57         .
```

```
## V58         .
## V59         .
## V60         .
## V61         .
## V62         .
## V63         .
## V64         .
## V65         .
## V66         .
## V67         .
## V68         4.020690e+00
## V69         .
## V70         .
## V71         .
## V72         .
## V73         .
## V74         .
## V75         .
## V76         .
## V77         .
## V78         .
## V79         .
## V80         .
## V81         .
## V82         1.461531e-04
## V83         .
## V84         .
## V85         .
## V86         .
## V87         .
## V88         .
## V89         .
## V90         .
## V91         .
```

The estimates for lambda.min are less sparse, but most of the coefficients are zero.

```
beta.hat.min = coef(G, s=CV$lambda.min)
beta.hat.min
```

```
## 92 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  0.5968106433
## V1           .
## V2           .
## V3           .
## V4           .
## V5           .
## V6           .
## V7           .
## V8           .
## V9           .
## V10          .
## V11          .
## V12          .
```

```
## V13          .
## V14          .
## V15          .
## V16           0.1669189168
## V17           1.1037582811
## V18          .
## V19           0.2367348558
## V20          .
## V21          .
## V22          .
## V23           0.7461633199
## V24          .
## V25          .
## V26          .
## V27           0.0819476459
## V28          .
## V29           0.0132221329
## V30           0.2740573279
## V31          -0.1150883168
## V32           0.2453629932
## V33          .
## V34          .
## V35          .
## V36          .
## V37          .
## V38          .
## V39          .
## V40          .
## V41           0.2204040865
## V42           0.0002254381
## V43          .
## V44          .
## V45          .
## V46          .
## V47          .
## V48          .
## V49          .
## V50          .
## V51          .
## V52          .
## V53          .
## V54           0.2856227196
## V55          .
## V56          .
## V57          .
## V58          .
## V59          .
## V60          .
## V61          .
## V62          .
## V63          .
## V64          .
## V65          .
## V66          .
```

```
## V67          0.1990739379
## V68          4.3127701244
## V69          0.0185885039
## V70          .
## V71          .
## V72          .
## V73          .
## V74          .
## V75          .
## V76          .
## V77          .
## V78          .
## V79          .
## V80          .
## V81          0.1364632270
## V82          0.2284211165
## V83          .
## V84          .
## V85          .
## V86          .
## V87          0.1845697105
## V88          .
## V89          .
## V90          .
## V91          .
```

3.Using the variables selected on the first half of the data, fit a model using lm on the second half of the data and report confidence intervals for the regression parameters in the model with the selected features. You can find the mutation names identified by position and amino acid here: http://stats191.stanford.edu/data/NRTI_muts.txt.

We use lambda.min as the optimal lambda.

```
beta.hat.min
```

```
## 92 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  0.5968106433
## V1           .
## V2           .
## V3           .
## V4           .
## V5           .
## V6           .
## V7           .
## V8           .
## V9           .
## V10          .
## V11          .
## V12          .
## V13          .
## V14          .
## V15          .
## V16          0.1669189168
## V17          1.1037582811
## V18          .
```

```
## V19          0.2367348558
## V20          .
## V21          .
## V22          .
## V23          0.7461633199
## V24          .
## V25          .
## V26          .
## V27          0.0819476459
## V28          .
## V29          0.0132221329
## V30          0.2740573279
## V31         -0.1150883168
## V32          0.2453629932
## V33          .
## V34          .
## V35          .
## V36          .
## V37          .
## V38          .
## V39          .
## V40          .
## V41          0.2204040865
## V42          0.0002254381
## V43          .
## V44          .
## V45          .
## V46          .
## V47          .
## V48          .
## V49          .
## V50          .
## V51          .
## V52          .
## V53          .
## V54          0.2856227196
## V55          .
## V56          .
## V57          .
## V58          .
## V59          .
## V60          .
## V61          .
## V62          .
## V63          .
## V64          .
## V65          .
## V66          .
## V67          0.1990739379
## V68          4.3127701244
## V69          0.0185885039
## V70          .
## V71          .
## V72          .
```

```
## V73          .
## V74          .
## V75          .
## V76          .
## V77          .
## V78          .
## V79          .
## V80          .
## V81          0.1364632270
## V82          0.2284211165
## V83          .
## V84          .
## V85          .
## V86          .
## V87          0.1845697105
## V88          .
## V89          .
## V90          .
## V91          .
```

```
subset <- c()
for (i in 2:nrow(beta.hat.min)) {
  if (beta.hat.min[i, 1] != 0) {
    subset <- c(subset, i-1)
  }
}
```

Fit the test model using lm

```
fit <- lm(test_Y ~ test_X[, subset])
summary(fit)
```

```
##
## Call:
## lm(formula = test_Y ~ test_X[, subset])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.00801 -0.26926 -0.06336  0.25921  2.62637
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)            0.312187   0.055786   5.596 4.97e-08 ***
## test_X[, subset]V16  0.006028   0.228306   0.026  0.97895
## test_X[, subset]V17  1.514206   0.187138   8.091 1.51e-14 ***
## test_X[, subset]V19  0.297160   0.072158   4.118 4.95e-05 ***
## test_X[, subset]V23  1.602078   0.215629   7.430 1.16e-12 ***
## test_X[, subset]V27 -0.120648   0.353916  -0.341  0.73342
## test_X[, subset]V29  0.492394   0.259679   1.896  0.05890 .
## test_X[, subset]V30  0.161084   0.383782   0.420  0.67499
## test_X[, subset]V31 -0.153514   0.090183  -1.702  0.08975 .
## test_X[, subset]V32  1.240631   0.219670   5.648 3.79e-08 ***
## test_X[, subset]V41  0.798910   0.339601   2.352  0.01930 *
## test_X[, subset]V42  0.848723   0.515602   1.646  0.10080
## test_X[, subset]V54 -0.054984   0.394619  -0.139  0.88928
```

```
## test_X[, subset]V67  0.139449   0.091653   1.521  0.12920
## test_X[, subset]V68  4.448945   0.061776  72.017  < 2e-16 ***
## test_X[, subset]V69  0.246080   0.132204   1.861  0.06368 .
## test_X[, subset]V81  0.436443   0.103818   4.204 3.47e-05 ***
## test_X[, subset]V82  0.431489   0.074149   5.819 1.53e-08 ***
## test_X[, subset]V87  0.590860   0.222929   2.650  0.00847 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5203 on 298 degrees of freedom
## Multiple R-squared:  0.9536, Adjusted R-squared:  0.9508
## F-statistic: 340.5 on 18 and 298 DF,  p-value: < 2.2e-16
```

The confidence intervals for the regression parameters in the model with the selected features:

```
confint(fit)
```

```
##                           2.5 %     97.5 %
## (Intercept)          0.20240119 0.42197182
## test_X[, subset]V16 -0.44326783 0.45532296
## test_X[, subset]V17  1.14592578 1.88248649
## test_X[, subset]V19  0.15515735 0.43916360
## test_X[, subset]V23  1.17772836 2.02642693
## test_X[, subset]V27 -0.81713898 0.57584268
## test_X[, subset]V29 -0.01864185 1.00343046
## test_X[, subset]V30 -0.59418326 0.91635087
## test_X[, subset]V31 -0.33098900 0.02396187
## test_X[, subset]V32  0.80832867 1.67293246
## test_X[, subset]V41  0.13059055 1.46722919
## test_X[, subset]V42 -0.16595970 1.86340545
## test_X[, subset]V54 -0.83157793 0.72160969
## test_X[, subset]V67 -0.04092036 0.31981745
## test_X[, subset]V68  4.32737223 4.57051712
## test_X[, subset]V69 -0.01409219 0.50625300
## test_X[, subset]V81  0.23213304 0.64075326
## test_X[, subset]V82  0.28556696 0.57741145
## test_X[, subset]V87  0.15214474 1.02957465
```

## Question 2.

In this question we will use the same data generating function from Q.5 of Assignment 4, i.e. a noisy version of lpsa of data(prostate) with k=20 junk features. Below we will ask for k=50 junk features as well.

1. Generate noise as in Q.5 of Assignment 4 with 20 junk features. Randomly split the data in half.

```
set.seed(0)
prostate = read.table("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data", header=T
data(prostate)
```

```
## Warning in data(prostate): data set 'prostate' not found
```

```
head(prostate)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
##   train
## 1  TRUE
## 2  TRUE
## 3  TRUE
## 4  TRUE
## 5  TRUE
## 6  TRUE
```

```
fun <- function(fit, k){
  matrix <- model.matrix(fit)
  n <- nrow(matrix)
  for (i in 1:k) {
    matrix <- cbind(matrix, rnorm(n))
  }
  matrix <- matrix[,-1]
  return(as.data.frame(matrix))
}
```

```
fit <- lm(lpsa ~ ., data = prostate)

new.prostate <- fun(fit, 20)
fit1 <- lm(lpsa ~ . - train, data = prostate)
var <- var(fit1$fitted.values)
noise = function(n) {
  return(rnorm(n, mean = 0, sd = sqrt(var/2)))
}

n <- nrow(new.prostate)

#new.prostate$lpsa <- prostate$lpsa + noise(n)
lpsa.noisy <- prostate$lpsa + noise(n)
head(new.prostate)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45 trainTRUE
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0         1
```

```
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0          1
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20          1
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0          1
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0          1
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0          1
##          V10        V11         V12         V13        V14         V15
## 1  1.2629543  1.4559884  0.96079238  1.63464429 -0.1802571  0.62015220
## 2 -0.3262334  0.2290196  1.79048505 -1.80832758  0.6850148  0.09990307
## 3  1.3297993  0.9965439 -1.06416516 -0.21388595  3.2664145  1.80770349
## 4  1.2724293  0.7818592  0.01763655  0.07036614  0.5606005 -1.50242998
## 5  0.4146414 -0.7767766 -0.38990863  0.54970767 -0.0690173  0.28564047
## 6 -1.5399500 -0.6159899 -0.49083275 -0.69682355 -0.9724429  0.84570696
##          V16        V17        V18        V19         V20        V21        V22
## 1 -0.4439419 -1.2886532 -1.3094299 -1.4012512 -0.04908307  0.7885664  0.3601695
## 2 -0.3125704  1.4191023 -0.7066913  1.2483585  0.79415451 -0.3420634  1.1094797
## 3 -0.6030044  1.3078278  1.0338917 -0.2470782 -1.41402352  1.8238489 -1.3062149
## 4 -1.0939347 -1.8049758  1.9727610  0.2455595 -1.78992736 -0.1481405 -1.8880233
## 5  0.7147062 -0.4840695  0.6875251 -0.7787236  1.35439733 -0.9715659  0.7937032
## 6 -0.1088123 -0.3732118  0.7382425 -1.7908862 -0.74776671 -0.3891390 -0.3846382
##           V23         V24         V25          V26         V27        V28
## 1 -0.06979488  0.20125990  0.30607300 -1.212218365  1.07181054 -0.8264229
## 2 -0.38085608  1.99883064 -0.56952568 -0.772288901  0.88423224  0.9878405
## 3  0.39355657 -0.05707633 -0.96340516 -0.350078968 -0.09899057  0.1707206
## 4  1.26555627  0.80354015  2.36194881  1.681213501 -2.04188166 -0.7351829
## 5  1.43107831 -0.04412127 -0.01870401 -2.290655613  2.05374640  0.1825896
## 6  0.17597035  1.53452555  0.29771328 -0.004754307  0.61485858  0.8219756
##          V29
## 1 -0.7300803
## 2  1.4560385
## 3  1.0485058
## 4  0.7994737
## 5 -0.5627331
## 6  0.1181808
```

Randomly split the data in half:

```r
sample <- sample.int(n = nrow(new.prostate), size = floor(0.5*nrow(new.prostate)), replace = F)

train.X <- as.matrix(new.prostate[sample, ])

train.lpsa.noisy <- as.matrix(lpsa.noisy[sample])

test.X <- as.matrix(new.prostate[-sample, ])
test.lpsa.noisy <- as.matrix(lpsa.noisy[-sample])
```

2. Using the first half of the data: fit the LASSO with parameter lambda.1se as selected by cv.glmnet, store the coefficients in a vector beta.lasso; do the same but for ridge regression storing the result in beta.ridge.

For Lasso

```r
cv_junk <- cv.glmnet(train.X, train.lpsa.noisy)
lambda1 <- cv_junk$lambda.1se

lasso_best <- glmnet(train.X , train.lpsa.noisy, alpha = 1, lambda = lambda1)
summary(lasso_best)
```

```
##           Length Class     Mode
## a0         1     -none-    numeric
## beta      29     dgCMatrix S4
## df         1     -none-    numeric
## dim        2     -none-    numeric
## lambda     1     -none-    numeric
## dev.ratio  1     -none-    numeric
## nulldev    1     -none-    numeric
## npasses    1     -none-    numeric
## jerr       1     -none-    numeric
## offset     1     -none-    logical
## call       5     -none-    call
## nobs       1     -none-    numeric
```

```r
beta.lasso <- coef(lasso_best)
beta.lasso
```

```
## 30 x 1 sparse Matrix of class "dgCMatrix"
##                    s0
## (Intercept) 2.2109505
## lcavol      0.2206026
## lweight     .
## age         .
## lbph        .
## svi         .
## lcp         .
## gleason     .
## pgg45       .
## trainTRUE   .
## V10         .
## V11         .
## V12         .
## V13         .
## V14         .
## V15         .
## V16         .
## V17         .
## V18         .
## V19         .
## V20         .
## V21         .
## V22         .
## V23         .
## V24         .
## V25         .
## V26         .
## V27         .
## V28         .
## V29         .
```

For Ridge regression

```r
ridge_best <- glmnet(train.X, train.lpsa.noisy, alpha = 0, lambda= lambda1)
beta.ridge <- coef(ridge_best)
beta.ridge
```

```
## 30 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept)  0.617293817
## lcavol       0.348956090
## lweight      0.311638088
## age          0.004656069
## lbph        -0.073327365
## svi          0.407949749
## lcp          0.094817709
## gleason      0.009638923
## pgg45       -0.003837962
## trainTRUE   -0.079384501
## V10          0.082182710
## V11          0.036697247
## V12          0.232617467
## V13          0.109850298
## V14         -0.030769585
## V15         -0.146535749
## V16          0.106110409
## V17         -0.084022649
## V18         -0.022868995
## V19          0.070417416
## V20          0.024251882
## V21          0.017018910
## V22          0.026628291
## V23          0.008783786
## V24         -0.034338201
## V25          0.104217632
## V26         -0.049945588
## V27          0.046081758
## V28          0.075633107
## V29         -0.061593255
```

3. Evaluate how well beta.lasso and beta.ridge predict on the second half of the data using mean squared error. Which one has smaller mean-squared error? (Answer will depend somewhat on the seed you use.)

```
lasso.test = predict(lasso_best, newx = test.X)
sum((lasso.test - test.lpsa.noisy)^2)
```

```
## [1] 96.53356
```

```
ridge.test = predict(ridge_best, newx = test.X)
sum((ridge.test - test.lpsa.noisy)^2)
```

```
## [1] 84.87737
```

If we use seed 0, we get 96.53356 for lasso MSE, and 84.87737 for ridge MSE. Therefore the Ridge has the smaller mean-squared sum for k=20 junk features.

4. Repeat steps 1.-3. using k=50 junk features.

4.1. Generate noise as in Q.5 of Assignment 4 with 50 junk features. Randomly split the data in half.

```
set.seed(0)
prostate = read.table("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data", header=T
data(prostate)
```

```
## Warning in data(prostate): data set 'prostate' not found
```

```r
head(prostate)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
##   train
## 1  TRUE
## 2  TRUE
## 3  TRUE
## 4  TRUE
## 5  TRUE
## 6  TRUE
```

```r
fun <- function(fit, k){
  matrix <- model.matrix(fit)
  n <- nrow(matrix)
  for (i in 1:k) {
    matrix <- cbind(matrix, rnorm(n))
  }
  matrix <- matrix[,-1]
  return(as.data.frame(matrix))
}
```

```r
fit <- lm(lpsa ~ ., data = prostate)

new.prostate <- fun(fit, 50)
fit1 <- lm(lpsa ~ . - train, data = prostate)
var <- var(fit1$fitted.values)
noise = function(n) {
  return(rnorm(n, mean = 0, sd = sqrt(var/2)))
}

n <- nrow(new.prostate)

#new.prostate$lpsa <- prostate$lpsa + noise(n)
lpsa.noisy <- prostate$lpsa + noise(n)
head(new.prostate)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45 trainTRUE
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0         1
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0         1
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20         1
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0         1
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0         1
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0         1
##          V10        V11         V12         V13        V14         V15
## 1  1.2629543  1.4559884  0.96079238  1.63464429 -0.1802571  0.62015220
## 2 -0.3262334  0.2290196  1.79048505 -1.80832758  0.6850148  0.09990307
## 3  1.3297993  0.9965439 -1.06416516 -0.21388595  3.2664145  1.80770349
## 4  1.2724293  0.7818592  0.01763655  0.07036614  0.5606005 -1.50242998
## 5  0.4146414 -0.7767766 -0.38990863  0.54970767 -0.0690173  0.28564047
```

```
## 6 -1.5399500 -0.6159899 -0.49083275 -0.69682355 -0.9724429  0.84570696
##          V16        V17        V18        V19         V20        V21        V22
## 1 -0.4439419 -1.2886532 -1.3094299 -1.4012512 -0.04908307  0.7885664  0.3601695
## 2 -0.3125704  1.4191023 -0.7066913  1.2483585  0.79415451 -0.3420634  1.1094797
## 3 -0.6030044  1.3078278  1.0338917 -0.2470782 -1.41402352  1.8238489 -1.3062149
## 4 -1.0939347 -1.8049758  1.9727610  0.2455595 -1.78992736 -0.1481405 -1.8880233
## 5  0.7147062 -0.4840695  0.6875251 -0.7787236  1.35439733 -0.9715659  0.7937032
## 6 -0.1088123 -0.3732118  0.7382425 -1.7908862 -0.74776671 -0.3891390 -0.3846382
##           V23         V24         V25          V26         V27        V28
## 1 -0.06979488  0.20125990  0.30607300 -1.212218365  1.07181054 -0.8264229
## 2 -0.38085608  1.99883064 -0.56952568 -0.772288901  0.88423224  0.9878405
## 3  0.39355657 -0.05707633 -0.96340516 -0.350078968 -0.09899057  0.1707206
## 4  1.26555627  0.80354015  2.36194881  1.681213501 -2.04188166 -0.7351829
## 5  1.43107831 -0.04412127 -0.01870401 -2.290655613  2.05374640  0.1825896
## 6  0.17597035  1.53452555  0.29771328 -0.004754307  0.61485858  0.8219756
##          V29         V30         V31         V32          V33        V34
## 1 -0.7300803 -3.04536393 -1.49254820 -1.79476514  0.059448309  0.3882388
## 2  1.4560385 -0.09738839 -0.13871324  0.58823384 -1.150067152  0.9521395
## 3  1.0485058 -1.29678356  0.59253975 -1.57257215  0.004480675 -0.7594887
## 4  0.7994737  0.19118007 -0.03356126  1.02309789 -0.883642641  0.4425276
## 5 -0.5627331  0.81111893 -0.20740668  0.07592384  0.671046073 -0.1234991
## 6  0.1181808 -0.73500227 -0.19082436 -0.11145924  0.004894727 -0.7082868
##          V35         V36        V37        V38         V39         V40
## 1 -0.3339082 -1.30871678  0.8510761 -0.8595847 -0.51358501 -1.35085334
## 2  1.2851482 -0.10788553  1.0259254  0.5004161 -0.48805032 -0.58233209
## 3 -0.1053090  1.24511470  0.3162624  0.1385575  0.02510549  0.79654213
## 4 -0.1149089  0.44791202 -0.7275867 -0.4437397  1.42108875 -0.06344118
## 5 -0.4192991 -0.38584056 -0.5529085 -0.6006577  0.52485306 -0.25846585
## 6 -1.2610495  0.05670356 -0.3633374  0.7056949  1.99220899  0.46347920
##           V41        V42        V43         V44        V45        V46
## 1 -0.45880691  0.6648860  1.0958040 -0.06274447  0.7623712  1.2040362
## 2  1.92815520 -0.4017993 -0.6960554  1.17554481  0.4161414 -0.8638468
## 3  1.35906060  2.6379403  1.3751865  1.64067952 -0.7693196 -1.0327522
## 4  0.44365925  0.3755638 -2.3718098 -0.75288293  0.4244615 -0.2547527
## 5 -1.25529071 -1.6100451 -0.2257597  0.35976347 -1.1834492  1.3238434
## 6  0.01766918 -0.2596963  0.1441632 -0.34031381 -0.1249676 -0.7655911
##          V47        V48        V49        V50         V51          V52
## 1 -0.4592783  0.5230101  1.7224750  1.2643219  0.01643097 -1.339440189
## 2  1.4970179  1.0769205  1.5101639 -1.1362568  0.10651692  0.002432616
## 3  0.2558538 -0.1208716 -0.4095998 -0.5242607  0.22364007  0.973897970
## 4 -0.6624787  0.1333708 -1.3070025 -0.5565111 -0.29902212  1.072541193
## 5  1.5406423  2.3013996  1.2131955 -1.9880002  0.84813507  0.311716722
## 6  1.7517668  1.7159142 -0.1640219 -0.1214389  1.49352275  1.903375819
##           V53        V54        V55        V56        V57          V58
## 1 -0.39764882  0.2395724  0.1272182  0.7268084  0.3314313 -0.921963759
## 2 -0.03909116 -0.5517267  1.6654661  0.8924211 -0.5292329  0.007477979
## 3 -2.17027739  1.9431638  1.2924916  0.4091263  0.5923591  1.543032182
## 4 -1.71775308 -0.5296448 -0.4042405  0.8830831 -0.5465008  0.173378092
## 5 -0.39470951 -1.0888619  1.0869762  0.8680603 -0.1067421  0.222274523
## 6 -0.85245380  1.6650690 -0.5207828 -1.8915226  1.8244807 -0.033634138
##          V59
## 1 -0.8551114
## 2  1.6990683
## 3 -0.7178294
```

```
## 4  0.1962242
## 5 -0.7635886
## 6 -0.3498082
```

Randomly split the data in half:

```
sample <- sample.int(n = nrow(new.prostate), size = floor(0.5*nrow(new.prostate)), replace = F)

train.X <- as.matrix(new.prostate[sample, ])

train.lpsa.noisy <- as.matrix(lpsa.noisy[sample])

test.X <- as.matrix(new.prostate[-sample, ])
test.lpsa.noisy <- as.matrix(lpsa.noisy[-sample])
```

4.2. Using the first half of the data: fit the LASSO with parameter lambda.1se as selected by cv.glmnet, store the coefficients in a vector beta.lasso; do the same but for ridge regression storing the result in beta.ridge.

For Lasso

```
cv_junk <- cv.glmnet(train.X, train.lpsa.noisy)
lambda1 <- cv_junk$lambda.1se

lasso_best <- glmnet(train.X , train.lpsa.noisy, alpha = 1, lambda = lambda1)
summary(lasso_best)
```

```
##           Length Class     Mode
## a0         1     -none-    numeric
## beta      59     dgCMatrix S4
## df         1     -none-    numeric
## dim        2     -none-    numeric
## lambda     1     -none-    numeric
## dev.ratio  1     -none-    numeric
## nulldev    1     -none-    numeric
## npasses    1     -none-    numeric
## jerr       1     -none-    numeric
## offset     1     -none-    logical
## call       5     -none-    call
## nobs       1     -none-    numeric
```

```
beta.lasso <- coef(lasso_best)
beta.lasso
```

```
## 60 x 1 sparse Matrix of class "dgCMatrix"
##                    s0
## (Intercept) 2.0517792
## lcavol      0.3342557
## lweight     .
## age         .
## lbph        .
## svi         .
## lcp         .
## gleason     .
## pgg45       .
## trainTRUE   .
## V10         .
## V11         .
```

```
## V12          .
## V13          .
## V14          .
## V15          .
## V16          .
## V17          .
## V18          .
## V19          .
## V20          .
## V21          .
## V22          .
## V23          .
## V24          .
## V25          .
## V26          .
## V27          .
## V28          .
## V29          .
## V30          .
## V31          .
## V32          .
## V33          .
## V34          .
## V35          .
## V36          .
## V37          .
## V38          .
## V39          .
## V40          .
## V41          .
## V42          .
## V43          .
## V44          .
## V45          .
## V46          .
## V47          .
## V48          .
## V49          .
## V50          .
## V51          .
## V52          .
## V53          .
## V54          .
## V55          .
## V56          .
## V57          .
## V58          .
## V59          .
```

For Ridge regression

```
ridge_best <- glmnet(train.X, train.lpsa.noisy, alpha = 0, lambda= lambda1)
beta.ridge <- coef(ridge_best)
beta.ridge
```

```
## 60 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -0.0374025092
## lcavol       0.2672598664
## lweight      0.3487003344
## age          0.0071626096
## lbph        -0.0214590428
## svi          0.4331439700
## lcp          0.0651910133
## gleason      0.0158625641
## pgg45        0.0045521161
## trainTRUE    0.1229327399
## V10         -0.0124696257
## V11         -0.0061167765
## V12          0.0183031290
## V13         -0.0004967506
## V14         -0.1053438560
## V15          0.0199629927
## V16         -0.0763808244
## V17          0.0023417026
## V18         -0.1680567937
## V19         -0.0763334038
## V20          0.0028460367
## V21          0.0681431836
## V22         -0.1604063892
## V23         -0.0532456521
## V24         -0.0915272683
## V25         -0.1640093779
## V26         -0.0596368320
## V27          0.0093064801
## V28          0.0950024528
## V29         -0.0606946150
## V30         -0.0339094942
## V31         -0.2200290798
## V32         -0.0397699263
## V33          0.0615982414
## V34         -0.0130694984
## V35          0.1703631436
## V36          0.0654487559
## V37          0.1013967799
## V38          0.0607132980
## V39         -0.1852128163
## V40         -0.1109864781
## V41         -0.1364378884
## V42         -0.0005347847
## V43          0.1406396607
## V44         -0.1668881226
## V45         -0.0867562485
## V46         -0.1123790899
## V47         -0.0120134787
## V48         -0.0413822898
## V49         -0.0408380390
## V50          0.1132903364
## V51         -0.1471950717
```

```
## V52         -0.1171854955
## V53          0.0815547404
## V54          0.2218051194
## V55         -0.1582894909
## V56         -0.0257772114
## V57         -0.0007842104
## V58         -0.0489221602
## V59         -0.0141857110
```

4.3. Evaluate how well beta.lasso and beta.ridge predict on the second half of the data using mean squared error. Which one has smaller mean-squared error? (Answer will depend somewhat on the seed you use.)

```r
lasso.test = predict(lasso_best, newx = test.X)
sum((lasso.test - test.lpsa.noisy)^2)
```

```
## [1] 55.49604
```

```r
ridge.test = predict(ridge_best, newx = test.X)
sum((ridge.test - test.lpsa.noisy)^2)
```

```
## [1] 79.8315
```

If we use seed 0, we get 55.49604 for lasso MSE, and 79.8315 for ridge MSE. Therefore the Lasso regression has the smaller mean-squared sum for k=50 junk features.

## Question 3.

1. Fit a logistic regression, modeling the probability of having any O-ring failures based on the temperature of the launch. Interpret the coefficients in terms of odds ratios.

```
orings= read.table('http://stats191.stanford.edu/data/Orings.table', header=TRUE, sep='')
head(orings)
```

```
##   Damaged Temp
## 1       1   53
## 2       1   57
## 3       1   58
## 4       1   63
## 5       0   66
## 6       0   67
```

```
fit <- glm(Damaged ~ Temp, family = binomial(), data = orings)
summary(fit)
```

```
##
## Call:
## glm(formula = Damaged ~ Temp, family = binomial(), data = orings)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0611  -0.7613  -0.3783   0.4524   2.2175
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  15.0429     7.3786   2.039   0.0415 *
## Temp         -0.2322     0.1082  -2.145   0.0320 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 28.267  on 22  degrees of freedom
## Residual deviance: 20.315  on 21  degrees of freedom
## AIC: 24.315
##
## Number of Fisher Scoring iterations: 5
```

```
exp(coef(fit))
```

```
##  (Intercept)         Temp
## 3.412315e+06 7.928171e-01
```

exponentiating the coefficients will give odd ratios.

2. From the fitted model, find the probability of an O-ring failure when the temperature at launch was 31 degrees. This was the temperature forecast for the day of the launching of the fatal Challenger flight on January 20, 1986.

```
logodds = predict(fit, list(Damaged = 1, Temp = 31), type='link')
logodds
```

```
##        1
## 7.845857
```

```r
prob = exp(logodds)/(1+exp(logodds))
prob
```

```
##         1
## 0.9996088
```

The probability is 99.96%

3. Find an approximate 95% confidence interval for the coefficient of temperature in the logistic regression using both the summary and confint. Are the confidence intervals the same? Why or why not?

The interval using Confint function

```r
confint(fit)[2,]
```

```
## Waiting for profiling to be done...
```

```
##       2.5 %      97.5 %
## -0.51547175 -0.06082076
```

The interval using R summary

```r
center = coef(fit)['Temp']
SE = sqrt(vcov(fit)['Temp', 'Temp'])
U = center + SE * qnorm(0.975)
L = center - SE * qnorm(0.975)
data.frame(L, U)
```

```
##               L           U
## Temp -0.4443022 -0.02002324
```

The profile intervals are not the same as default intervals because it is calculated using a large sample size.

## Question 4.

Since NETREV is a linear combination of the other covariates PCREV, NSAL, and FEXP, we drop the NETREV column.

```
health <- read.table("http://www1.aucegypt.edu/faculty/hadi/RABE5/Data5/P014.txt", header = TRUE, sep =
head(health)
```

```
##   RURAL BED MCDAYS TDAYS PCREV NSAL FEXP NETREV
## 1     0 244    128   385 23521 5230 5334  12957
## 2     1  59    155   203  9160 2459  493   6208
## 3     0 120    281   392 21900 6304 6115   9481
## 4     0 120    291   419 22354 6590 6346   9418
## 5     0 120    238   363 17421 5362 6225   5834
## 6     1  65    180   234 10531 3622  449   6460
```

```
health <- health[,1:7]
head(health)
```

```
##   RURAL BED MCDAYS TDAYS PCREV NSAL FEXP
## 1     0 244    128   385 23521 5230 5334
## 2     1  59    155   203  9160 2459  493
## 3     0 120    281   392 21900 6304 6115
## 4     0 120    291   419 22354 6590 6346
## 5     0 120    238   363 17421 5362 6225
## 6     1  65    180   234 10531 3622  449
```

1. Using a logistic regression model, test the null hypothesis that the measured covariates have no power to distinguish between rural facilities and than non-rural facilities. Use level $\alpha$=0.05

```
null <- glm(RURAL ~ 1, data= health, family = binomial())
full <- glm(RURAL ~ BED + MCDAYS + TDAYS + PCREV + NSAL + FEXP, data = health, family = binomial())
summary(full)
```

```
##
## Call:
## glm(formula = RURAL ~ BED + MCDAYS + TDAYS + PCREV + NSAL + FEXP,
##     family = binomial(), data = health)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0352  -0.6100   0.4801   0.7529   1.4173
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.736e+00  1.379e+00   2.710  0.00673 **
## BED         -3.182e-02  2.829e-02  -1.125  0.26072
## MCDAYS       1.585e-02  9.325e-03   1.700  0.08908 .
## TDAYS       -6.694e-03  9.399e-03  -0.712  0.47635
## PCREV        5.293e-05  1.263e-04   0.419  0.67507
## NSAL        -7.146e-04  3.284e-04  -2.176  0.02955 *
## FEXP         2.932e-04  2.629e-04   1.115  0.26471
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 67.083  on 51  degrees of freedom
## Residual deviance: 48.809  on 45  degrees of freedom
## AIC: 62.809
##
## Number of Fisher Scoring iterations: 5
```

```r
1 - pchisq(67.083 - 48.809, 51 -45)
```

```
## [1] 0.005582724
```

The P-value for the null hypothesis that the measured covariates have no power to distinguish between rural facilities and than non-rural facilities is 0.005582724 ($< \alpha = 0.05$). So we reject the null hypothesis.

2.Use a model selection technique based on AIC to choose a model that seems to best describe the outcome RURAL based on the measured covariates.

```r
library(MASS)
step(full, direction='both', scope=list(upper= ~., lower = ~ 1), trace =FALSE, k = 2)
```

```
##
## Call:  glm(formula = RURAL ~ BED + MCDAYS + NSAL + FEXP, family = binomial(),
##     data = health)
##
## Coefficients:
## (Intercept)          BED       MCDAYS         NSAL         FEXP
##   3.6442709   -0.0366403    0.0126199   -0.0007526    0.0003439
##
## Degrees of Freedom: 51 Total (i.e. Null);  47 Residual
## Null Deviance:       67.08
## Residual Deviance: 49.36     AIC: 59.36
```

3. Repeat 2. but using BIC instead. Is the model the same?

```r
step(full, direction='both', scope=list(upper= ~., lower = ~ 1), trace =FALSE, k = log(nrow(health)))
```

```
##
## Call:  glm(formula = RURAL ~ NSAL, family = binomial(), data = health)
##
## Coefficients:
## (Intercept)         NSAL
##   3.3126144   -0.0006671
##
## Degrees of Freedom: 51 Total (i.e. Null);  50 Residual
## Null Deviance:       67.08
## Residual Deviance: 55.42     AIC: 59.42
```

The models aren't the same. 4. Report estimates of the parameters for the variables in your final model. How are these to be interpreted?

```r
coef(step(full, direction='both', scope=list(upper= ~., lower = ~ 1), trace =FALSE, k = 2))
```

```
##   (Intercept)           BED        MCDAYS          NSAL          FEXP
##  3.6442708758 -0.0366403079  0.0126198692 -0.0007525902  0.0003439185
```

5. Report confidence intervals for the parameters in 4. Do you think you can trust these intervals?

```r
confint(step(full, direction='both', scope=list(upper= ~., lower = ~ 1), trace =FALSE, k = 2))
```

```
## Waiting for profiling to be done...
```

```
##                      2.5 %        97.5 %
```

```
## (Intercept)  1.3230177562  6.5725174611
## BED         -0.0842320869  0.0011069442
## MCDAYS       0.0003627587  0.0280625335
## NSAL        -0.0014713073 -0.0002003883
## FEXP        -0.0001150223  0.0008951408
```

Most of the intervals have upper and lower bounds that are very close to zero, meaning these covariates are almost have no effect on explaining RURAL.