Homework 3

Phoebe Parrish

2/14/2020

Homework 3

Question 1

For this problem, you will use the `iris` data. You will apply K-nearest neighbors to classify the observations.

(A) Make a plot showing a range of values of K in K-nearest neighbors, ranging from $K = 1$ to $K = n/2$, on the x-axis. On this plot, display five curves:

- The training error rate
- The test error rate
- The test error rate estimated using leave-one-out cross-validation
- The test error rate estimated using 5-fold cross-validation
- The test error rate estimated using 10-fold cross-validation

For this problem please implement cross-validation yourself, without using an existing `R` function.

(B) Comment on the plot obtained in (A). Which value of K results in the lowest estimated test error?

Functions to calculate KNN training error and cross-validation test error:

```r
set.seed(123)

# function to estimate KNN training error; returns training error
knn.train.err <- function(df, k){
  # split data into training and test sets
  df.mod <- rowid_to_column(df, "ID")
  train <- df.mod %>% sample_frac(0.75)
  test <- anti_join(df.mod, train, by="ID")

  # fit a KNN model
  knn.model <- knn(train[,c("Sepal.Length", "Petal.Length")],
                   train[,c("Sepal.Length", "Petal.Length")],
                   train$Species, k=k)
  out.df <- data.frame(train, Predicted=knn.model) %>%
    mutate(correct.call = ifelse(Species==Predicted, 1,0))
  head(out.df)

  # calculate training error
  train.err <- (1-(sum(out.df$correct.call/nrow(out.df))))
  return(train.err)
}

# function to do validation set cross-validation; returns test error
val.set.cv <- function(df, k){
  # split data into training and test sets
  df.mod <- rowid_to_column(df, "ID")
  train <- df.mod %>% sample_frac(0.5) # split data in half
  test <- anti_join(df.mod, train, by="ID")

  # fit a KNN model
  knn.model <- knn(train[,c("Sepal.Length", "Petal.Length")],
                   test[,c("Sepal.Length", "Petal.Length")],
                   train$Species, k=k)
  out.df <- data.frame(test, Predicted=knn.model) %>%
    mutate(correct.call = ifelse(Species==Predicted, 1,0))
  head(out.df)

  # calculate test error
  test.err <- (1-(sum(out.df$correct.call/nrow(out.df))))
  return(test.err)
}

# function to do leave-one-out cross-validation; returns test error
LOO.cv <- function(df, k){
  # calculate number of rounds of C-V to do (based on length of df)
  n <- nrow(df)

  # make an empty vector to store test error from each round of LOO C-V
  test.errors <- c()

  for (i in 1:n){

    # split data into training and test sets
    df.mod <- rowid_to_column(df, "ID")
    test <- df.mod %>% filter(ID==i)
    train <- anti_join(df.mod, test, by="ID")

    # fit the model to the training data, get prediction for the test set (length=1)
    knn.model <- knn(train[,c("Sepal.Length", "Petal.Length")],
                     test[,c("Sepal.Length", "Petal.Length")],
                     train$Species, k=k)
    out.df <- data.frame(test, Predicted=knn.model) %>%
      mutate(correct.call = ifelse(Species == Predicted, 1, 0))

    # save whether the output was correct in a vector
    test.errors <- c(test.errors, out.df$correct.call)
  }

  # calculate LOO C-V test error
  LOO.test.err <- 1-(sum(test.errors)/length(test.errors))
  return(LOO.test.err)
}
```

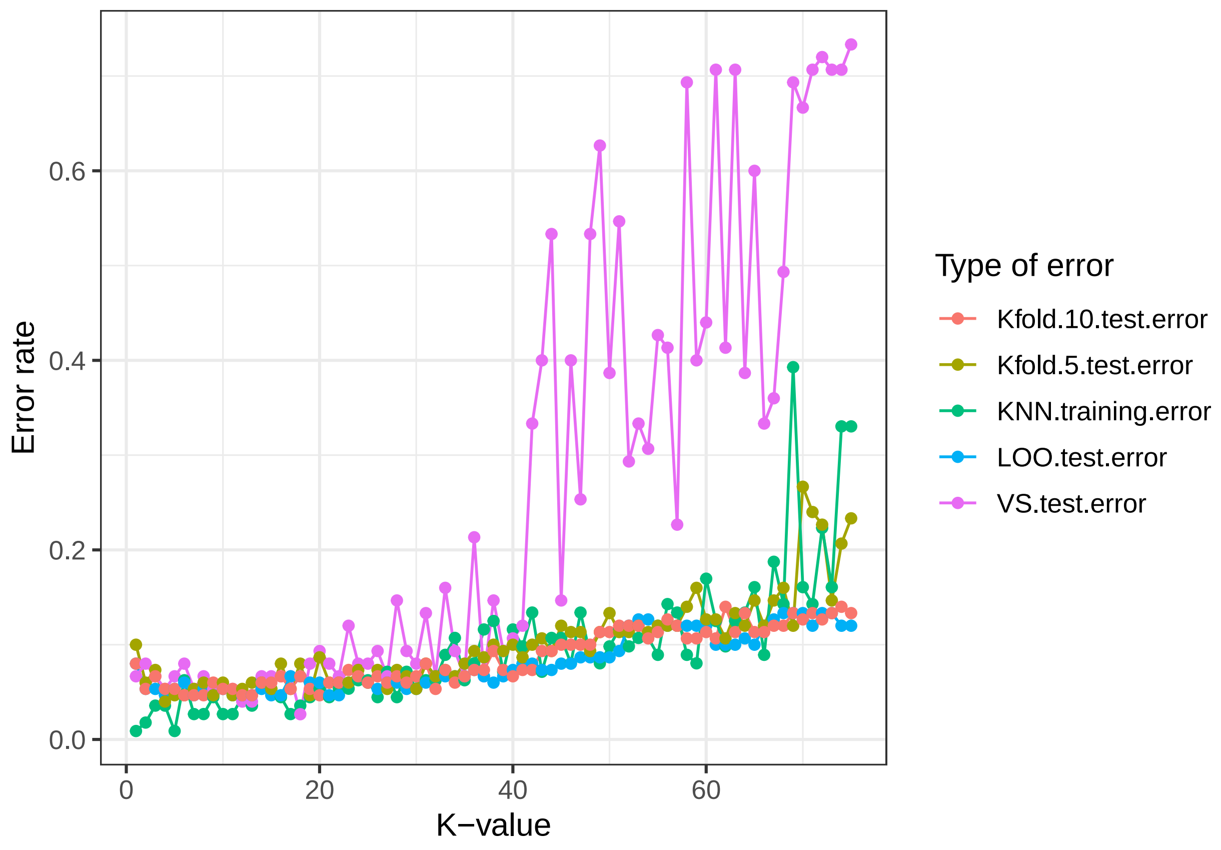
```r
# define a function to do K-fold cross-validation with inputs:
#  (1) df: dataframe
#  (2) knn.k: k-value to be used for KNN
#  (3) cv.k: k-value to be used for K-fold C-V
#  which returns test error
Kfold.cv <- function(df, knn.k, cv.k){
  n <- nrow(df)

  # randomize the dataset to avoid selecting only one class
  #  in the training/test sets
  df.random <- df[sample(1:nrow(df)),]

  # calculate the number of tests to run (n/k)
  n.test <- n/cv.k

  # generate start and end coordinates for each fold
  start.set <- seq(1, n, by=n.test)
  end.set <- seq(n.test, n, by=n.test)

  # make an empty vector to store the test errors
  test.errors <- c()

  for(i in 1:cv.k){
    # get start and end coordinates
    test.start <- start.set[i]
    test.end <- end.set[i]

    # split into training and test sets
    df.mod <- rowid_to_column(df.random, "ID")
    test <- df.mod %>% filter(ID >= test.start & ID <= test.end)
    train <- anti_join(df.mod, test, by="ID")

    # fit the KNN model to the training data, get predictions for the test data
    knn.model <- knn(train[,c("Sepal.Length", "Petal.Length")],
                     test[,c("Sepal.Length", "Petal.Length")],
                     train$Species, k=knn.k)
      out.df <- data.frame(test, Predicted=knn.model) %>%
      mutate(correct.call = ifelse(Species == Predicted, 1, 0))

    # calculate test error
    test.err <- (1-(sum(out.df$correct.call)/nrow(out.df)))
    test.errors <- c(test.errors, test.err)
  }

  # get the mean test error across all K folds
  Kfold.test.err <- mean(test.errors)
  return(Kfold.test.err)
}
```

Here I implement the KNN and C-V functions:

```r
len.iris <- nrow(iris)

# define a set of K-vals
```

```r
k.vals <- seq(1,(len.iris/2))

# define empty vectors to store training/test errors
train.errors <- c()
vs.test.errors <- c()
LOO.test.errors <- c()
Kfold5.test.errors <- c()
Kfold10.test.errors <- c()

# calculate training error and estimate C-V test errors for each KNN k-value
for(i in k.vals){
  # define dataset to be used for cross-validation
  df = iris

  # calculate KNN training error
  train.err <- knn.train.err(df, i)
  train.errors <- c(train.errors, train.err)

  # estimate C-V method training errors:
  vs.test.err <- val.set.cv(df, i)
  vs.test.errors <- c(vs.test.errors, vs.test.err)

  LOO.test.err <- LOO.cv(df, i)
  LOO.test.errors <- c(LOO.test.errors, LOO.test.err)

  Kfold5.test.err <- Kfold.cv(df, i, 5)
  Kfold5.test.errors <- c(Kfold5.test.errors, Kfold5.test.err)

  Kfold10.test.err <- Kfold.cv(df, i, 10)
  Kfold10.test.errors <- c(Kfold10.test.errors, Kfold10.test.err)
}
```

I then bound the errors and KNN K-values into a dataframe and plotted the results:



(B)

```
# get the K-value(s) associated with the minimum test error for each C-V method
min_k.VS <- top_n(cv.error.df, -1, VS.test.error)$K.value

min_k.LOO <- top_n(cv.error.df, -1, LOO.test.error)$K.value
min_k.LOO <- paste(min_k.LOO, collapse=",")

min_k.Kfold5 <- top_n(cv.error.df, -1, Kfold.5.test.error)$K.value
min_k.Kfold5 <- paste(min_k.Kfold5, collapse=",")

min_k.Kfold10 <- top_n(cv.error.df, -1, Kfold.10.test.error)$K.value
min_k.Kfold10 <- paste(min_k.Kfold10, collapse=",")
```

I then stored these in an output dataframe:

type.of.error	minimum.k.vals
VS.test.error	18
LOO.test.error	4,7,9,11,12,13,15,16,21,22
Kfold.5.test.error	4
Kfold.10.test.error	6,7,8,12,13,20

The value of K that results in the lowest estimated test error depends on the cross-validation method used. Some methods had ties for lowest test error between multiple K-values. Refer to the table above for the precise results; overall, it seems that K-values between 2 and 14 resulted in the lowest estimated test error for each method.

Question 2

On Q4 of HW2, you fit a bunch of models to the `Auto` dataset from the `ISLR` library. Now, use cross-validation to estimate the test errors for several linear models on this dataset. Also, compute the training error for each model.

Make a table listing the models that you considered, as well as the training error and estimated test error for each model. Which of the models that you considered had the lowest estimated test error?

For this problem, you can implement cross-validation yourself, or you can use an `R` function that performs cross-validation for you.

I first adapted my K-fold C-V model from above to work with linear regression (rather than K-nearest neighbors):

```r
set.seed(123)

# define a function to do K-fold cross-validation with inputs:
#  (1) df: dataframe
#  (2) cv.k: k-value to be used for K-fold C-V
#  which returns test error
Kfold.cv.lm <- function(df, cv.k){
  n <- nrow(df)

  # randomize the dataset to avoid selecting only one class
  #  in the training/test sets
  df.random <- df[sample(1:nrow(df)),]

  # calculate the number of tests to run (n/k)
  n.test <- n/cv.k

  # generate start and end coordinates for each fold
  start.set <- seq(1, n, by=n.test)
  end.set <- seq(n.test, n, by=n.test)

  # make empty vectors to store the training and test errors
  train.errors <- c()
  test.errors <- c()

  for(i in 1:cv.k){
    # get start and end coordinates
    test.start <- start.set[i]
    test.end <- end.set[i]

    # split into training and test sets
    df.mod <- rowid_to_column(df.random, "ID")
    test <- df.mod %>% filter(ID >= test.start & ID <= test.end)
    train <- anti_join(df.mod, test, by="ID")

    # drop ID columns to avoid confusion during lm fitting
    test <- test %>% dplyr::select(-ID)
    train <- train %>% dplyr::select(-ID)
    # fit the KNN model to the training data, get predictions for the test data
    cv.lm <- lm(mpg ~ horsepower, data=train)
    cv.lm.test.pred <- predict(cv.lm, test)
    cv.lm.test.pred.df <- tibble("y.test"=test$mpg, "y.hat.test"=cv.lm.test.pred) %>%
      mutate("diff"=y.test-y.hat.test)

    # calculate test error
    test.err <- (1/n.test)*sum((cv.lm.test.pred.df$diff)^2)
    test.errors <- c(test.errors, test.err)

    # calculate training error
    train.err <- (1/(n-n.test))*sum((resid(cv.lm))^2)
    train.errors <- c(train.errors, train.err)
  }

  # get the mean test and training error across all K folds, save in a list
  Kfold.test.err <- mean(test.errors)
  Kfold.train.err <- mean(train.errors)
  results <- list("train.error"=Kfold.train.err, "test.error"=Kfold.test.err)

  # return a list of training and test error
  return(results)
}
```

Here, I calculate the training error and estimated test error for three linear models using K-fold cross-validation:

```
set.seed(123)

# make a dataframe for each model and run K-fold C-V on each:
auto.linear.df <- Auto %>% dplyr::select(c(mpg, horsepower))
linear.cv.fit <- Kfold.cv.lm(auto.linear.df, 8)

auto.log.df <- Auto %>% dplyr::select(c(mpg, horsepower)) %>%
  mutate(horsepower=log(horsepower))
log.cv.fit <- Kfold.cv.lm(auto.log.df, 8)

poly.horsepower <- poly(Auto$horsepower, degree=2)
mpg <- Auto$mpg
auto.poly.df <- tibble("mpg"=mpg, "horsepower"=poly.horsepower)
poly.cv.fit <- Kfold.cv.lm(auto.poly.df, 8)
```

I then saved the K-fold C-V results in a dataframe:

model	training.error	test.error
linear	23.91339	24.40096
log	20.13439	20.43023
polynomial	18.97501	19.13132

In this analysis, the polynomial model had the lowest estimated test error.

Question 3

In this problem, we'll see a (very!!) simple simulated example where a least squares linear model is "too flexible".

(A) First, generate some data with $n = 100$ and $p = 10,000$ features, and a quantitative response. Write out an expression for the model corresponding to this data generation procedure.

(B) What is the value of the irreducible error?

(C) Consider a very simple model-fitting procedure that just predicts 0 for every observation. That is, $\hat{f}(x) = 0$ for all x. * What is the bias of this procedure? * What is the variance of this procedure? * What is the expected prediction error of this procedure? * Use the validation set approach to estimate the test error of this procedure. * Do the expected prediction error and the test error agree with each other? Explain.

(D) Now use the validation set approach to estimate the test error of a least squares linear model using $X_1 ... X_10000$ to predict Y. What is the estimated test error?

(E) Comment on your answers to (C) and (D). Which of the two procedures has a smaller estimated test error? higher bias? higher variance? In answering this question, be sure to think carefully about how the data were generated.

(A)

```r
set.seed(123)

y <- rnorm(100)
x <- matrix(rnorm(10000*100), ncol=10000)

df.sim <- data.frame(y, x)
```

The equation for this model would be $Y = \epsilon$ because the y values were not generated based on the x values.

(B) The irreducible error is the variance of the difference between each y value and the mean of all y values.

```r
set.seed(123)

y.diff <- (y - mean(y))
irred.error <- var(y.diff)
cat("Irreducible error =", round(irred.error,4))
```

```
## Irreducible error = 0.8332
```

(C)
(i) The bias is the difference between our prediction and the true mean.

```r
set.seed(123)

mean.y <- mean(y)
cat("Mean y from my model =", round(mean.y, 4))
```

```
## Mean y from my model = 0.0904
```

So here the bias would be: $0 - 0.0904 = -0.0904$.

(ii) The variance of this model should be zero, because we predict that y=0 for every value of x.

(iii) The expected prediction error would be the bias of our model plus the irreducible error. In this case, that would be:

```r
set.seed(123)

exp.pred.err <- (-0.0904)^2 + irred.error
cat("Expected prediction error =", round(exp.pred.err, 4))
```

```
## Expected prediction error = 0.8414
```

(iv)

```r
set.seed(123)

df.sim.mod <- rowid_to_column(df.sim, "ID")
train.sim <- df.sim.mod %>% sample_frac(0.5) # split data in half
test.sim <- anti_join(df.sim.mod, train.sim, by="ID")

# calculate model test error based on the MSE (where y^=0 for each row)
mse.lm.sim <- mean((test.sim$y-0)^2)
cat("Predicted test error =", round(mse.lm.sim,4))
```

```
## Predicted test error = 0.7846
```

(v) My answers for (iii) and (iv) are fairly close, although the validation set error was slightly lower than my expected prediction error.

(D)

```r
set.seed(123)

df.sim.mod <- rowid_to_column(df.sim, "ID")
train.sim <- df.sim.mod %>% sample_frac(0.5) # split data in half
test.sim <- anti_join(df.sim.mod, train.sim, by="ID")

# fit a linear model
lm.sim <- lm(y ~ ., data=train.sim)
test.sim <- test.sim %>% mutate(lm.predict = predict(lm.sim, test.sim))

mse.lm.sim <- mean((test.sim$y-test.sim$lm.predict)^2)

cat("Estimated test error =", mse.lm.sim)
```

```
## Estimated test error = 218.4698
```

(E) The **variance** for the model in part (C) is lower than that from part (D) because in (C) we predict $\hat{f}(x) = 0$ for every sample. In (D), on the other hand, our predictions change based on the x values and will therefore have higher variance.

The **bias** for the model in (C) is higher because this model will always give a slightly inaccurate prediction of the true population mean (which should equal zero) due to the difference between our sample mean and this true mean (which we could only achieve if we had an infinite sample size). The model in part (D), on the other hand, has lower bias because we are not systematically over- or underestimating the mean, we're just fitting it to every observation because there is no true relationship between X and Y.

The features for the model in part (D) are random, so fitting a linear regression model based on these random features should give a higher **test error** (since we are overfitting the data). On the other hand for part (C) our $\hat{f}(x) = 0$ model should be fairly accurate, because we sampled these data from a mean-zero normal distribution.

Question 4

In lecture, we discussed the "pitfalls of cross-validation". Here, we are going to continue to work with the simulated data from the previous problem, in order to illustrate the problem with Option 1.

(A) Calculate the correlation between each feature and the response. Make a histogram of these correlations. What are the values of the 10 largest absolute correlations?

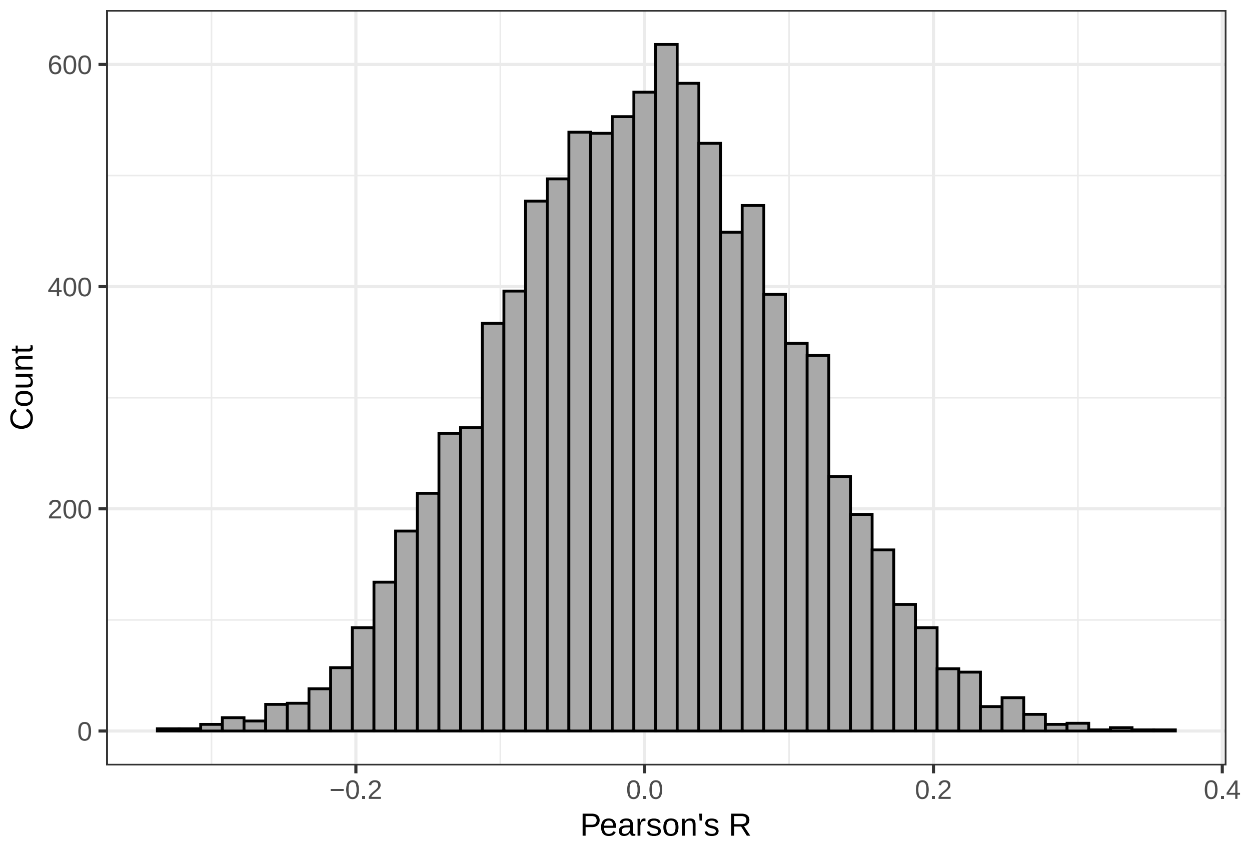
(B) Now try out Option 1 with $q = 10$ (to keep things simple, you can use the "validation set" version of Option 1). What is the estimated test error?

(C) Now try out Option 2 with $q = 10$ (to keep things simple, you can use the "validation set" version of Option 2). What is the estimated test error?

(D) Comment on your results in (C) and (C). How does this relate to the discussion of Option 1 versus Option 2 from lecture? Explain how you can see that Option 1 gave you a useless (i.e. misleading, inaccurate, wrong) estimate of the test error.

(A)

```r
cor.vals <- c()
x.vals <- seq(2,ncol(df.sim))

for(i in x.vals){
  y <- df.sim$y
  x <- df.sim[,i]
  correl <- cor(x, y)
  cor.vals <- c(cor.vals, correl)
}

cor.df <- tibble("pearsons.r"=cor.vals, "x.idx"=x.vals)
```



```
top.n <- top_n(cor.df, 10, abs(pearsons.r))

top.cor.df <- tibble("abs.pearsons.r"=abs(top.n$pearsons.r), "x.idx"=top.n$x.idx) %>%
  arrange(desc(abs.pearsons.r)) %>% mutate("rank"=seq(1,10)) %>% dplyr::select(rank, everything())

kable(top.cor.df)
```

rank	abs.pearsons.r	x.idx
1	0.3638350	4013
2	0.3416071	4976
3	0.3340142	3146
4	0.3321909	5901
5	0.3302984	5222
6	0.3243617	3450
7	0.3230033	7915
8	0.3199438	7035
9	0.3097622	7594
10	0.3076815	7404

(B) Option 1

```
set.seed(123)

# get rows corresponding to X values that are highly correlated with y
top.cor.df1 <- df.sim[,c(1, top.cor.df$x.idx)]
```

```r
# split into training and test sets
top.cor.mod1 <- rowid_to_column(top.cor.df1, "ID")
top.cor.train1 <- top.cor.mod1 %>% sample_frac(0.5)
top.cor.test1 <- anti_join(top.cor.mod1, top.cor.train1, by="ID")

# fit the model to the training data, predict test data values
top.cor.lm1 <- lm(y ~ ., data=top.cor.train1)
test.predict <- predict(top.cor.lm1, top.cor.test1)

# calculate error
test.predict.df1 <- tibble(y=top.cor.test1$y, y.hat=test.predict) %>%
  mutate(error=(y-y.hat)^2)
val.set.error1 <- mean(test.predict.df1$error)
cat("Option 1 error =", val.set.error1)
```

```
## Option 1 error = 0.4287478
```

(C) Option 2

```r
set.seed(123)

# split into training and test sets
df.sim.mod2 <- rowid_to_column(df.sim, "ID")
df.sim.train <- df.sim.mod2 %>% sample_frac(0.5)
df.sim.test <- anti_join(df.sim.mod2, df.sim.train, by="ID")

# calculate correlation between y and each X in the training set, save as a dataframe
cor.vals2 <- c()
x.vals2 <- seq(3,ncol(df.sim))

for(i in x.vals2){
  y <- df.sim.train$y
  x <- df.sim.train[,i]
  correl <- cor(x, y)
  cor.vals2 <- c(cor.vals2, correl)
}

cor.df2 <- tibble("pearsons.r"=cor.vals2, "x.idx"=x.vals2)

# get top 10 correlations
top.n2 <- top_n(cor.df2, 10, abs(pearsons.r))
top.n2.df <- tibble("abs.pearsons.r"=abs(top.n2$pearsons.r), "x.idx"=top.n2$x.idx) %>%
  arrange(desc(abs.pearsons.r)) %>% mutate("rank"=seq(1,10)) %>%
  dplyr::select(rank, everything())

# extract columns containing X-values that are highly correlated with y from the dataset
top.cor.df2 <- df.sim.train[,c(2, top.n2.df$x.idx)]

# fit the model to the training data, predict values for the test data
top.cor.lm2 <- lm(y ~ ., data=top.cor.df2)
test.predict2 <- predict(top.cor.lm2, df.sim.test)

# calculate error
test.predict.df2 <- tibble(y=df.sim.train$y, y.hat=test.predict2)
val.set.error2 <- MSE(test.predict.df2$y.hat, test.predict.df2$y)
```

```r
cat("Option 2 error =", val.set.error2)
```

```
## Option 2 error = 1.108996
```

(D) My error for option 1 is lower because I am selecting the top correlated features based partly on the test data. This gives a falsely low estimate of the predicted test error because my model (which should ideally be generated using *only* the training data) has already "seen" some of the test data.

Question 5

In this problem, you will analyze a (real, not simulated) dataset of your choice with a quantitative response Y, and $p \geq 50$ quantitative predictors.

(A) Describe the data. Where did you get it from? What is the meaning of the response, and what are the meanings of the predictors?

(B) Fit a least squares linear model to the data, and provide an estimate of the test error. (Explain how you got this estimate.)

(C) Fit a ridge regression model to the data, with a range of values of the tuning parameter λ. Make a plot.

(D) What value of λ in the ridge regression model provides the smallest estimated test error? Report this estimate of test error. (Also, explain how you estimated test error.)

(E) Repeat (C), but for a lasso model.

(F) Repeat (D), but for a lasso model. Which features are included in this lasso model?

In this problem, you may use the function in the `glmnet` package that performs cross-validation.

(A) For this problem, I'm using the Communities and Crime dataset from the UCI Machine Learning Repository. This combines socio-economic data from the 1990 US census with data from a 1990 survey of law enforcement in the US and crime data collected by the FBI in 1995.

```r
# read in the crime dataset, set "?" to NA
crime = read.csv("../input_data/communities.csv", header = FALSE, na.strings="?", strip.white=TRUE)

# read in and assign column names for the crime dataset
names = read.csv("../input_data/attributes.csv", header=TRUE, strip.white=TRUE)
colnames(crime) = names$attributes

# drop first 5 variables, which are non-quantitative, as well as variables 81-127,
#  which contain a lot of missing data. Keep variable 128 to use as the response.
crime.df <- crime[,c(6:80,128)]
```

(B)

```r
set.seed(123)

# drop rows containing NA values
crime.df <- na.omit(crime.df)

# predict the number of violent crimes per 100K population using all
#  remaining features
crime.lm <- glm(ViolentCrimesPerPop ~ ., data=crime.df)

# run K-fold cross-validation using cv.glm from the boot package
```

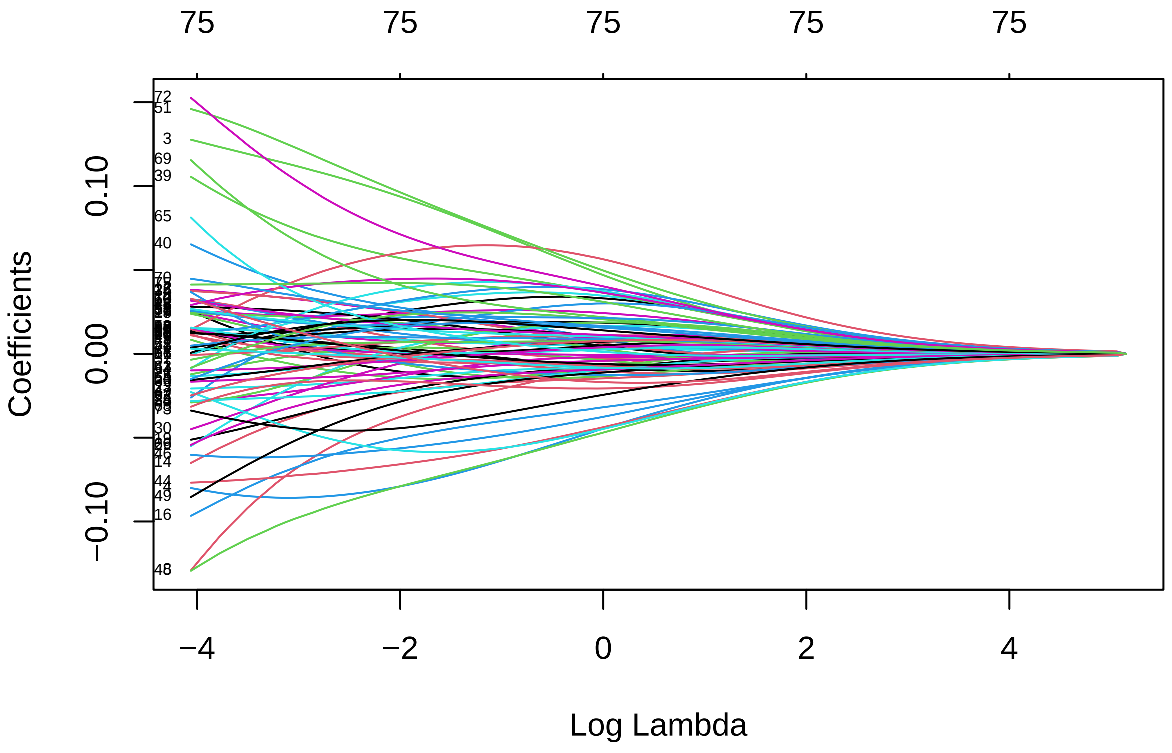
```
crime.lm.cv <- cv.glm(data=crime.df, glmfit=crime.lm, K=10)
crime.test.pred <- crime.lm.cv$delta[1]
cat("Predicted test error =", round(crime.test.pred, 5))
```

```
## Predicted test error = 0.01891
```

I obtained this estimate of the test error using K-fold cross-validation with K=10. I ran cross-validation using the cv.glm function from the R package boot, and the output here is the average test error of the 10 validation sets.

(C)

```
glmnet.x <- as.matrix(crime.df[,1:75])
glmnet.y <- crime.df$ViolentCrimesPerPop

ridge.fit <- glmnet(x=glmnet.x, y=glmnet.y, a=0)
plot(ridge.fit, xvar="lambda", label=TRUE)
```



(D)

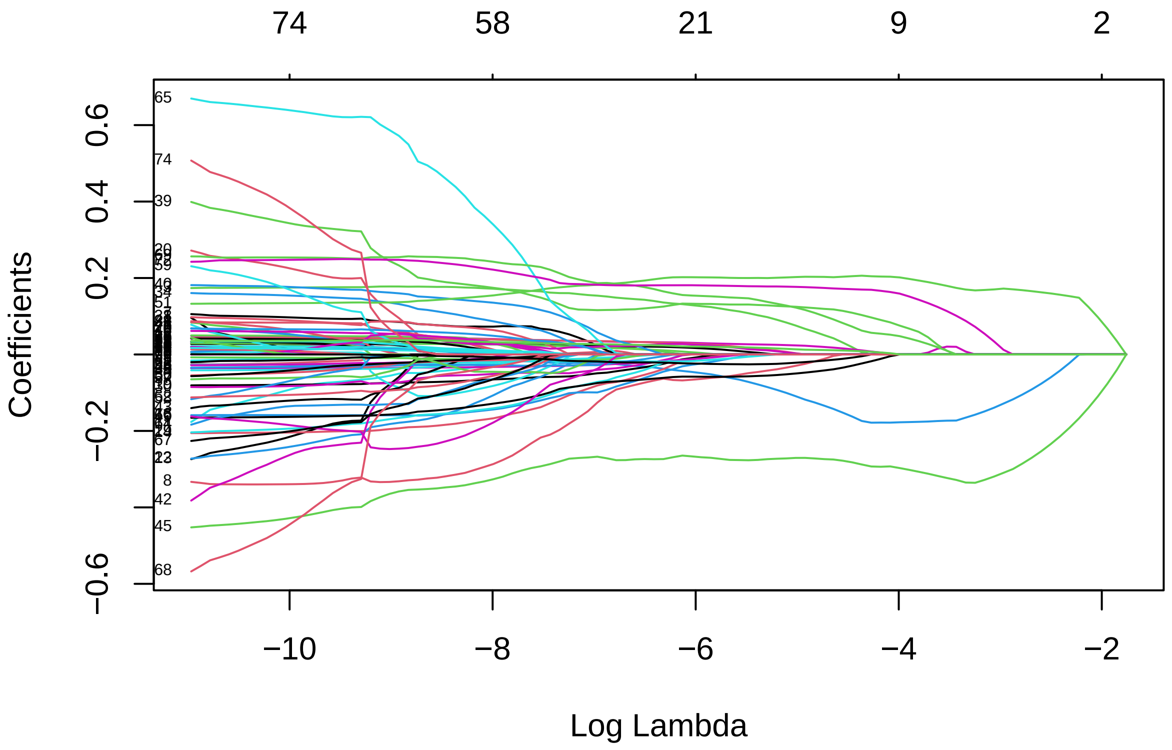
```
set.seed(123)

cv.ridge.fit <- cv.glmnet(x=glmnet.x, y=glmnet.y, alpha=0)

cat("lambda.min =", cv.ridge.fit$lambda.min)
cat("\nlog(lambda.min) =", log10(cv.ridge.fit$lambda.min))

ridge.error.min <- cv.ridge.fit$cvm[cv.ridge.fit$lambda == cv.ridge.fit$lambda.min]
cat("\nTest error at lambda.min =", ridge.error.min)
```

```
## lambda.min = 0.01722378
## log(lambda.min) = -1.763872
## Test error at lambda.min = 0.01904191
```

I obtained this λ that provides the smallest estimated test error using the cv.glmnet function with the alpha parameter set to 0 for ridge regression. This performs K-fold cross-validation (with K=10) across many values of lambda, as seen in the plot above.

(E)

```
lasso.fit <- glmnet(x=glmnet.x, y=glmnet.y, a=1)
plot(lasso.fit, xvar="lambda", label=TRUE)
```



(F)

```
set.seed(123)

cv.lasso.fit <- cv.glmnet(x=glmnet.x, y=glmnet.y, alpha=1)

cat("lambda.min =", cv.lasso.fit$lambda.min)
cat("\nlog(lambda.min) =", log10(cv.lasso.fit$lambda.min))

lasso.error.min <- cv.lasso.fit$cvm[cv.lasso.fit$lambda == cv.lasso.fit$lambda.min]
cat("\nTest error at lambda.min =", lasso.error.min)
```

```
## lambda.min = 0.0003710751
## log(lambda.min) = -3.430538
## Test error at lambda.min = 0.01887501
```

As in part (D), I obtained the λ with the smallest estimated test error using the cv.glmnet function with K=10 folds across many values of lambda. This time, however, the alpha parameter set to 1 for lasso regression.

```
# get non-zero coefficients from the lasso model
lasso.coef.mat <- coef(cv.lasso.fit, s= "lambda.min")
lasso.coef.df <- as.data.frame(as.matrix(lasso.coef.mat)) %>% rownames_to_column()
colnames(lasso.coef.df) <- c("coef.name", "coef.val")
```

```
lasso.coef.df <- lasso.coef.df %>% filter(coef.val != 0)
kable(lasso.coef.df)
```

coef.name	coef.val
(Intercept)	0.5305644
racepctblack	0.1702590
racePctWhite	-0.0162290
racePctHisp	0.0241219
agePct12t21	0.0740843
agePct12t29	-0.2759270
agePct16t24	0.0229496
numbUrban	-0.0231023
pctUrban	0.0398471
pctWWage	-0.1615707
pctWFarmSelf	0.0377664
pctWInvInc	-0.1405019
pctWPubAsst	0.0032026
pctWRetire	-0.0642934
medFamInc	0.0054681
whitePerCap	-0.0944248
blackPerCap	-0.0179712
indianPerCap	-0.0322400
AsianPerCap	0.0278338
OtherPerCap	0.0390002
HispPerCap	0.0257918
PctPopUnderPov	-0.1370659
PctLess9thGrade	-0.0491754
PctBSorMore	0.0597445
PctUnemployed	-0.0142577
PctEmploy	0.0819064
PctEmplManu	-0.0176731
PctEmplProfServ	-0.0032080
PctOccupManu	0.0002812
PctOccupMgmtProf	0.0029123
MalePctDivorce	0.1711125
MalePctNevMarr	0.1326362
FemalePctDiv	-0.0757642
PctKids2Par	-0.3196494
PctYoungKids2Par	-0.0282238
PctTeen2Par	0.0059248
PctWorkMomYoungKids	0.0286365
PctWorkMom	-0.1252149
NumIlleg	-0.0141074
PctIlleg	0.1576385
PctImmigRecent	0.0101985
PctImmigRec8	-0.0095143
PctRecImmig8	0.0082765
PctRecImmig10	0.0267832
PctNotSpeakEnglWell	-0.0503827
PctLargHouseFam	-0.0457233
PctLargHouseOccup	-0.0582727
PersPerOccupHous	0.3141215
PersPerOwnOccHous	-0.1643629

coef.name	coef.val
PersPerRentOccHous	-0.0561905
PctPersOwnOccup	-0.0283621
PctPersDenseHous	0.2387762
PctHousLess3BR	0.0464467
MedNumBR	0.0037013
HousVacant	0.2177059
PctHousOccup	-0.0074235
PctVacantBoarded	0.0315292

As seen in the table above, I have 60 remaining non-zero coefficients in my model. These include a number of statistics about the communities in the dataset, including percentages of different ages and racial/ethnic groups, socioeconomic data such as employment rates, and housing data. I would strongly recommend not over-interpreting the effects of different features on my model, given that there are likely many factors (including racial discrimination, xenophobia, sexism, and many more) that play into these statistics and/or impact crime rates which are not captured in this dataset. Also, given that much of the data are provided by law enforcement surveys/databases, the dataset is likely biased in many ways.