Stats 101C Homework 2

Charles Liu (304804942)

10/22/2020

Loading Necessary Packages:

```
library(MASS)
library(ISLR)
library(class)
```

Problem 1 (Exercise 4) **When the number of features p is large, there tends to be a deterioration in the performance of KNN and other local approaches that perform prediction using only observations that are near the test observation for which a prediction must be made. This phenomenon is known as the curse of dimensionality, and it ties into the fact that non-parametric approaches often perform poorly when p is large. We mensionality will now investigate this curse.**

(a) Suppose that we have a set of observations, each with measurements on p = 1 feature, X. We assume that X is uniformly (evenly) distributed on [0, 1]. Associated with each observation is a response value. Suppose that we wish to predict a test observation's response using only observations that are within 10% of the range of X closest to that test observation. For instance, in order to predict the response for a test observation with X = 0.6, we will use observations in the range [0.55, 0.65]. On average, what fraction of the available observations will we use to make the prediction?

ANSWER: Let's say a test observation is $X \in [X_a, X_b]$, and this will give you 10% of the observations because $X \sim \mathcal{U}\backslash\rangle\{\wr\nabla\Updownarrow(0,1)$. This results in the range of training observations as $[X_a - 0.05, X_b + 0.05]$ for the prediction. Overall the cases, we see that the fraction of the observations used for the prediction is **10%**.

(b) Now suppose that we have a set of observations, each with measurements on p = 2 features, X1 and X2. We assume that (X1,X2) are uniformly distributed on [0, 1] × [0, 1]. We wish to predict a test observation's response using only observations that are within 10% of the range of X1 and within 10% of the range of X2 closest to that test observation. For instance, in order to predict the response for a test observation with X1 = 0.6 and X2 = 0.35, we will use observations in the range [0.55, 0.65] for X1 and in the range [0.3, 0.4] for X2. On average, what fraction of the available observations will we use to make the prediction?

ANSWER: We see that this is similar to Exercise 4a, and we see that both X_1 and X_2 are *10%* of observations each. Now assuming that both cases are independent from each other, we can simply multiply the probabilities of these two to get our fraction used to make each prediction. Simple math shows us that $0.10 * 0.10 = 0.01$, and we say that the fraction of the available observations that we will use to make the prediction is **1%**.

(c) Now suppose that we have a set of observations on p = 100 features. Again the observations are uniformly distributed on each feature, and again each feature ranges in value from 0 to 1. We wish to predict a test observation's response using observations within the 10% of each feature's range that is closest to that test observation. What fraction of the available observations will we use to make the prediction?

ANSWER: We see that this is similar to Exercise 4b, and we are now using $p = 100$. Using the same reasoning and assumptions as Exercise 4b, we will simply multiply the 10% raised to the 100th power. To show our work, $0.10^{100} \approx 0$, and the fraction of the available observations that we will use to make the prediction approaches **0%**.

(d) Using your answers to parts (a)–(c), argue that a drawback of KNN when p is large is that there are very few training observations "near" any given test observation.

ANSWER: We see that as we increase our dimensionality (p), the probability of our training observations being near our test observations (X) approaches zero. We can depict this using a limit and shown as follows...

$$\lim_{p \to \infty} (0.10)^p = 0$$

The drawback to this is that whenever we use KNN with p being very large, the K chosen "nearest neighbors" will *NOT* actually be very close. This comes from the fact that there does not exist training observations that are "close"/"near" across all the dimensionalities of p.

(e) Now suppose that we wish to make a prediction for a test observation by creating a p-dimensional hypercube centered around the test observation that contains, on average, 10% of the training observations. For p = 1, 2, and 100, what is the length of each side of the hypercube? Comment on your answer.

Note: A hypercube is a generalization of a cube to an arbitrary number of dimensions. When p = 1, a hypercube is simply a line segment, when p = 2 it is a square, and when p = 100 it is a 100-dimensional cube.

ANSWERs BELOW:
1) When $p = 1$, we use the following to solve for the cube $(0.10)^{\frac{1}{p}}$, and we will find $(0.10)^{\frac{1}{1}} = 0.10$. Therefore, we see that our fraction is **10%**.
2) When $p = 2$, we use the following to solve for the cube $(0.10)^{\frac{1}{p}}$, and we will find $(0.10)^{\frac{1}{2}} = 0.316227766$.

Therefore, we see that our fraction is about **31.62%**.
3) When $p = 100$, we use the following to solve for the cube $(0.10)^{\frac{1}{p}}$, and we will find $(0.10)^{\frac{1}{100}} = 0.977237221$.
Therefore, we see that our fraction is **97.72%**.
NOTE: The hypercube needs to extend further for each of the dimensions in the training observations, so the observations used must be farther than the 10% of training observations used for prediction. As we increase our dimensionalities (p), the 10% of observations being used are *NOT* "close"/"near" at all.

Problem 2 (Exercise 8) Suppose that we take a data set, divide it into equally-sized training and test sets, and then try out two different classification procedures. First we use logistic regression and get an error rate of 20% on the training data and 30% on the test data. Next we use 1-nearest neighbors (i.e. K = 1) and get an average error rate (averaged over both test and training data sets) of 18%. Based on these results, which method should we prefer to use for classification of new observations? Why?

ANSWER: I would prefer to use the *Logistic Regression* approach. For $K = 1$ KNN, it will cause our data to be overfitted and gives a training data error rate of *0%*. This means our test data error rate will actually be *36%* ($2 * 18\% = 36\%$). Thus, this tells us the *Logistic Regression* is actually the better approach to use rather than using the KNN approach with $K = 1$.

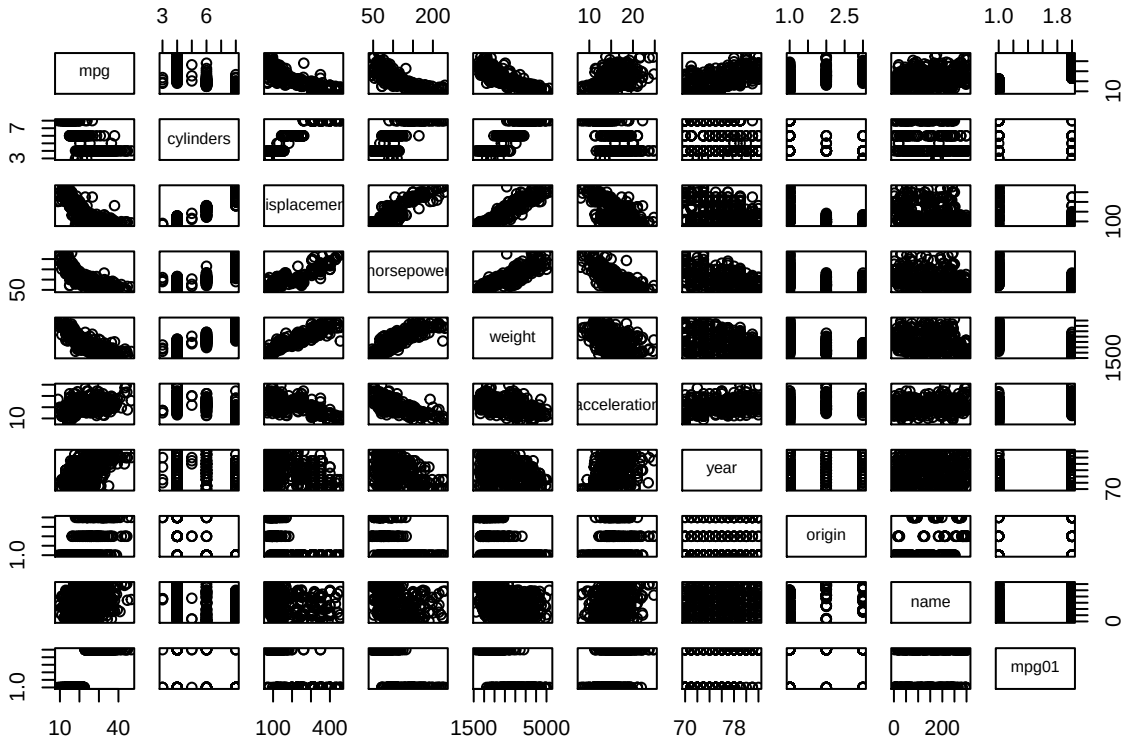
Problem 3 (Exercise 11) In this problem, you will develop a model to predict whether a given car gets high or low gas mileage based on the Auto data set.

(a) Create a binary variable, mpg01, that contains a 1 if mpg contains a value above its median, and a 0 if mpg contains a value below its median. You can compute the median using the median() function. Note: you may find it helpful to use the data.frame() function to create a single data set containing both mpg01 and the other Auto variables.

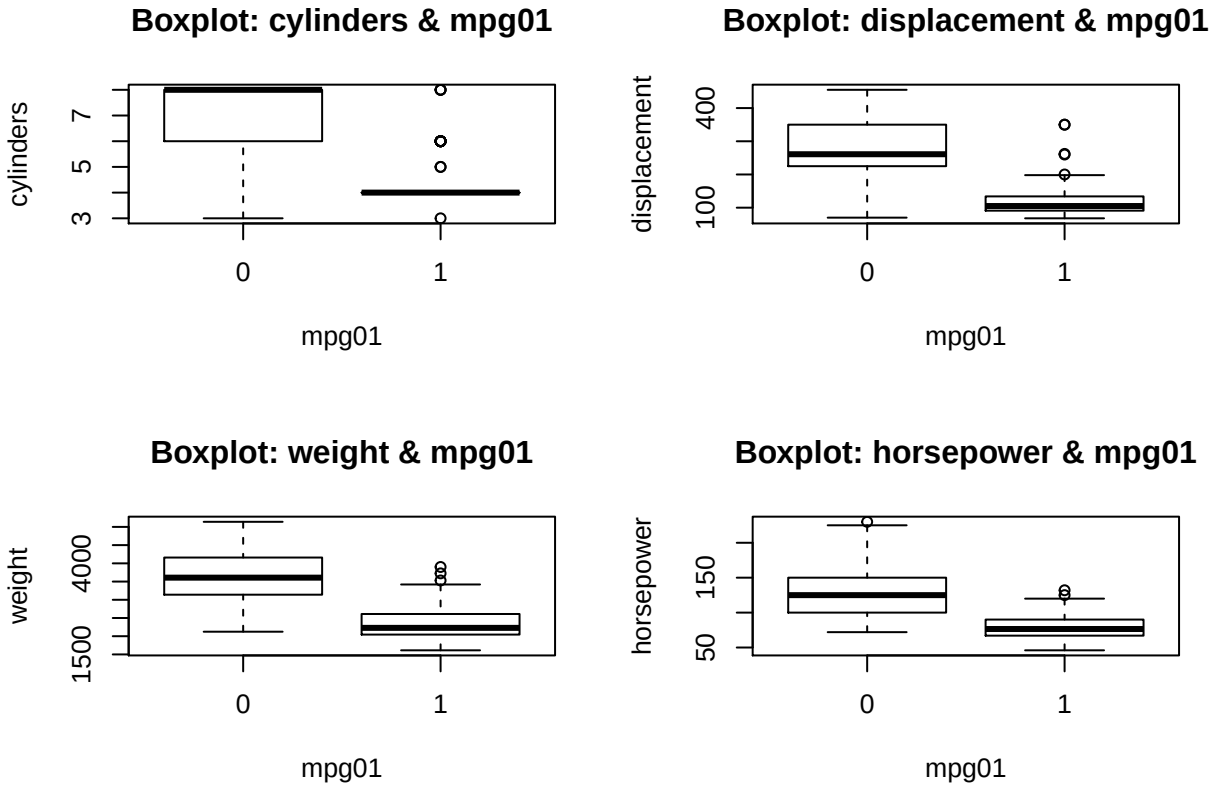
```
attach(Auto)
mpg01 <- factor(as.numeric(mpg > median(mpg)))
Auto_new_df <- data.frame(Auto, mpg01)
```

(b) Explore the data graphically in order to investigate the association between mpg01 and the other features. Which of the other features seem most likely to be useful in predicting mpg01? Scatterplots and boxplots may be useful tools to answer this question. Describe your findings.

```
# Scatterplots
pairs(Auto_new_df) # we see there is some relationship with "cylinders", "displacement", "weight", and
```




```
# Boxplots
par(mfrow = c(2, 2))
boxplot(cylinders ~ mpg01, main = "Boxplot: cylinders & mpg01")
boxplot(displacement ~ mpg01, main = "Boxplot: displacement & mpg01")
boxplot(weight ~ mpg01, main = "Boxplot: weight & mpg01")
boxplot(horsepower ~ mpg01, main = "Boxplot: horsepower & mpg01")
```



ANSWER: We see that "cylinders", "displacement", "weight", and "horsepower" all seem most likely to be predicting the "mpg01" variable.

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

```
# Using our "year" variable to split our data (since the dataset is huge, we can divide it by half)
train_data <- (year %% 2 == 0)

# Create our training for "mpg01"
mpg01_train <- mpg01[train_data]

# Create our test for "mpg01"
mpg01_test <- mpg01[!train_data]

# Create our training for Auto
Auto_train <- Auto[train_data, ]

# Create our test for Auto
Auto_test <- Auto[!train_data, ]
```

(d) Perform LDA on the training data in order to predict mpg01 using the variables that seemed most associated with mpg01 in (b). What is the test error of the model obtained?

```
# Create our LDA model
lda_model <- lda(mpg01 ~ cylinders + weight + displacement +
            horsepower, data = Auto, subset = train_data)

# Create our prediction from our model and testing data
lda_prediction <- predict(lda_model, Auto_test)

# Create the classifications
lda_prediction_class <- lda_prediction$class

# Create the Confusion Matrix with the classifications and "mpg01" test
lda_confusion_matrix <- table(lda_prediction_class, mpg01_test)
colnames(lda_confusion_matrix) <- c("FALSE", "TRUE")
rownames(lda_confusion_matrix) <- c("FALSE", "TRUE")
lda_confusion_matrix
```

```
##                     mpg01_test
## lda_prediction_class FALSE TRUE
##                FALSE    86    9
##                TRUE     14   73
```

```
# Finally, we need to find the test error rate
mean(lda_prediction_class != mpg01_test) # use mean function
```

```
## [1] 0.1263736
```

ANSWER: Our test error rate for the LDA model is 12.63736%.

(e) Perform QDA on the training data in order to predict mpg01 using the variables that seemed most associated with mpg01 in (b). What is the test error of the model obtained?

```
# Create our QDA model
qda_model <- qda(mpg01 ~ cylinders + weight + displacement +
            horsepower, data = Auto, subset = train_data)

# Create our prediction from our model and testing data
qda_prediction <- predict(qda_model, Auto_test)

# Create the classifications
qda_prediction_class <- qda_prediction$class

# Create the Confusion Matrix with the classifications and "mpg01" test
qda_confusion_matrix <- table(qda_prediction_class, mpg01_test)
colnames(qda_confusion_matrix) <- c("FALSE", "TRUE")
rownames(qda_confusion_matrix) <- c("FALSE", "TRUE")
qda_confusion_matrix
```

```
##                     mpg01_test
## qda_prediction_class FALSE TRUE
##                FALSE    89   13
```

```
##                TRUE     11   69
```

```
# Finally, we need to find the test error rate
mean(qda_prediction_class != mpg01_test) # use mean function
```

```
## [1] 0.1318681
```

ANSWER: Our test error rate for the QDA model is 13.18681%.

(f) Perform logistic regression on the training data in order to predict mpg01 using the variables that seemed most associated with mpg01 in (b). What is the test error of the model obtained?

```
# Create our GLM model (have to use a family, so we need Binomial)
glm_model <- glm(mpg01 ~ cylinders + weight + displacement +
            horsepower, data = Auto, subset = train_data, family = binomial)

# Create our prediction from our model and testing data (use type = "responses" because we are only loo
glm_prediction <- predict(glm_model, Auto_test, type = "response")

# Create our probability for the prediction
glm_prob_prediction <- rep(0, length(glm_prediction))
# If the response is greater than 0.50, then it is "TRUE = 1"
glm_prob_prediction[glm_prediction > 0.50] <- 1

# Create the Confusion Matrix with the classifications and "mpg01" test
glm_confusion_matrix <- table(glm_prob_prediction, mpg01_test)
colnames(glm_confusion_matrix) <- c("FALSE", "TRUE")
rownames(glm_confusion_matrix) <- c("FALSE", "TRUE")
glm_confusion_matrix
```

```
##                     mpg01_test
## glm_prob_prediction FALSE TRUE
##               FALSE    89   11
##               TRUE     11   71
```

```
# Finally, we need to find the test error rate
mean(glm_prob_prediction != mpg01_test) # use mean function
```

```
## [1] 0.1208791
```

ANSWER: Our test error rate for the GLM model is 12.08791%.

(g) Perform KNN on the training data, with several values of K, in order to predict mpg01. Use only the variables that seemed most associated with mpg01 in (b). What test errors do you obtain? Which value of K seems to perform the best on this data set?

```
# Create our training and testing data for KNN
KNN_train <- cbind(cylinders, weight, displacement, horsepower)[train_data, ]
KNN_test <- cbind(cylinders, weight, displacement, horsepower)[!train_data, ]

# Create our KNN from our training and testing data (we don't use the predict() function, isntead we us
KNN_prediction <- knn(KNN_train, KNN_test, mpg01_train, k = 1)
```

```r
# Create the Confusion Matrix with the classifications and "mpg01" test
KNN_confusion_matrix <- table(KNN_prediction, mpg01_test)
colnames(KNN_confusion_matrix) <- c("FALSE", "TRUE")
rownames(KNN_confusion_matrix) <- c("FALSE", "TRUE")
KNN_confusion_matrix
```

```
##               mpg01_test
## KNN_prediction FALSE TRUE
##          FALSE    83   11
##          TRUE     17   71
```

```r
# Finally, we need to find the test error rate
mean(KNN_prediction != mpg01_test) # use mean function
```

```
## [1] 0.1538462
```

ANSWER: Our test error rate for the KNN model is 15.38462%