Data Mining and Statistical Learning

Homework 3

Introduction

In machine learning, classification refers to a set of modeling problems where a class label is assigned to a given input data. There are many different types of classification methods, but for the purpose of this homework, we will only be focused mainly on binary classification. Binary classification refers to predicting one of two classes. In this homework, we will be working with the **auto mpg** dataset. This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. The dataset was used in the 1983 American Statistical Association Exposition.

The goal of this homework is to use several classifiers in order to predict mpg using the variables that seemed most associated with it

Problem Statement and Data Set

The objective of our analysis is to predict whether a given car gets high or low gas mileage based 7 car attributes such as cylinders, displacement, horsepower, weight, acceleration, model year and origin.

The dataset used in the assignment is a slightly modified version of the dataset provided in the StatLib library. The data is a collection of 398 automobile records from 1970 to 1982 and concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multi-valued discrete and 5 continuous attributes." (Quinlan, 1993).

The modified dataset has undergone some data preprocessing. For the purpose of simplicity in this homework, rows with missing values were removed. Also the last column (car names), which is text/string was also removed. A new binary response variable was created **mpg01** that contains a 1 if mpg contains a value above its median, and a 0 if mpg contains a value below its median. The new cleaned data contains 392 observations and 8 columns. The dataset attribute information is described below:

1. mpg01: Miles per gallon (binary variable)
2. cylinders: A chamber where fuel is combusted and power is generated (multi-valued discrete)

3. displacement: Determined by calculating the engine cylinder bore area multiplied by the stroke of the crankshaft, and then multiplied by the number of cylinders (continuous)
4. horsepower: The power an engine produces (continuous)
5. weight: Mass of the car (continuous)
6. acceleration: The rate of change of velocity as a function of time (continuous)
7. year: model year (1970 through 1982) (multi-valued discrete)
8. origin: 1 is a car made in the United States, 2 in Europe and 3 in Asia or other part of the world (multi-valued discrete)

Exploratory Data Analysis

For any real dataset, we first need to conduct empirical data analysis to have a better understanding of the data. We will first explore the summary statistics of the full data. The two tables below show the minimum, 25th quantile, median, mean, 75th quantile, and maximum values of each variable in the Auto MPG dataset.

Table 1: Summary statistics 1

mpg01	cylinders	displacement	horsepower
Min. :0.0	Min. :3.000	Min. : 68.0	Min. : 46.0
1st Qu.:0.0	1st Qu.:4.000	1st Qu.:105.0	1st Qu.: 75.0
Median :0.5	Median :4.000	Median :151.0	Median : 93.5
Mean :0.5	Mean :5.472	Mean :194.4	Mean :104.5
3rd Qu.:1.0	3rd Qu.:8.000	3rd Qu.:275.8	3rd Qu.:126.0
Max. :1.0	Max. :8.000	Max. :455.0	Max. :230.0

Table 2: Summary statistics 2

weight	acceleration	year	origin
Min. :1613	Min. : 8.00	Min. :70.00	Min. :1.000
1st Qu.:2225	1st Qu.:13.78	1st Qu.:73.00	1st Qu.:1.000
Median :2804	Median :15.50	Median :76.00	Median :1.000
Mean :2978	Mean :15.54	Mean :75.98	Mean :1.577

weight	acceleration	year	origin
3rd Qu.:3615	3rd Qu.:17.02	3rd Qu.:79.00	3rd Qu.:2.000
Max. :5140	Max. :24.80	Max. :82.00	Max. :3.000

From the summary statistics, we see that cylinders vary from 3 to 8, while horsepower is ranging from 46 through 230. The mass of the vehicles is between a range of 1613 and 5140 (could be Kg or Lb).

Another integral part of EDA is the histogram plot. Histograms are good for showing general distributional features of the variables.

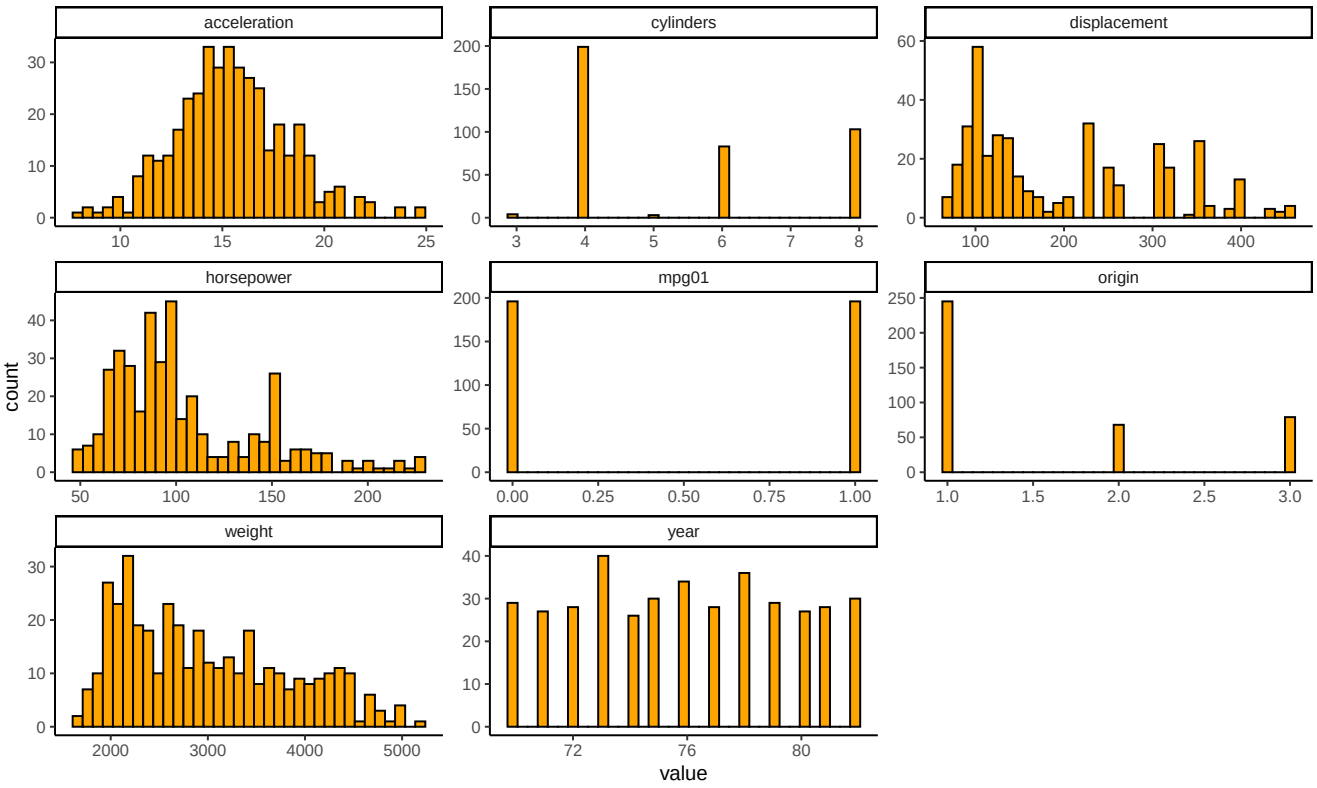


Figure 1: Histogram of each column in the Auto mpg dataset

We can see roughly where the peaks of the distribution are, whether the distribution is skewed or symmetric, and if there are any outliers. The histograms show that most of the cars in the dataset are American made with 4 or 8 cylinders. The acceleration does somewhat follow a normal distribution.

To have a clear picture of which of the other features seem most likely to be useful in predicting mpg01, we can use a pairs plot which would allow us to see both distribution of single variables and relationships between two variables (see below)

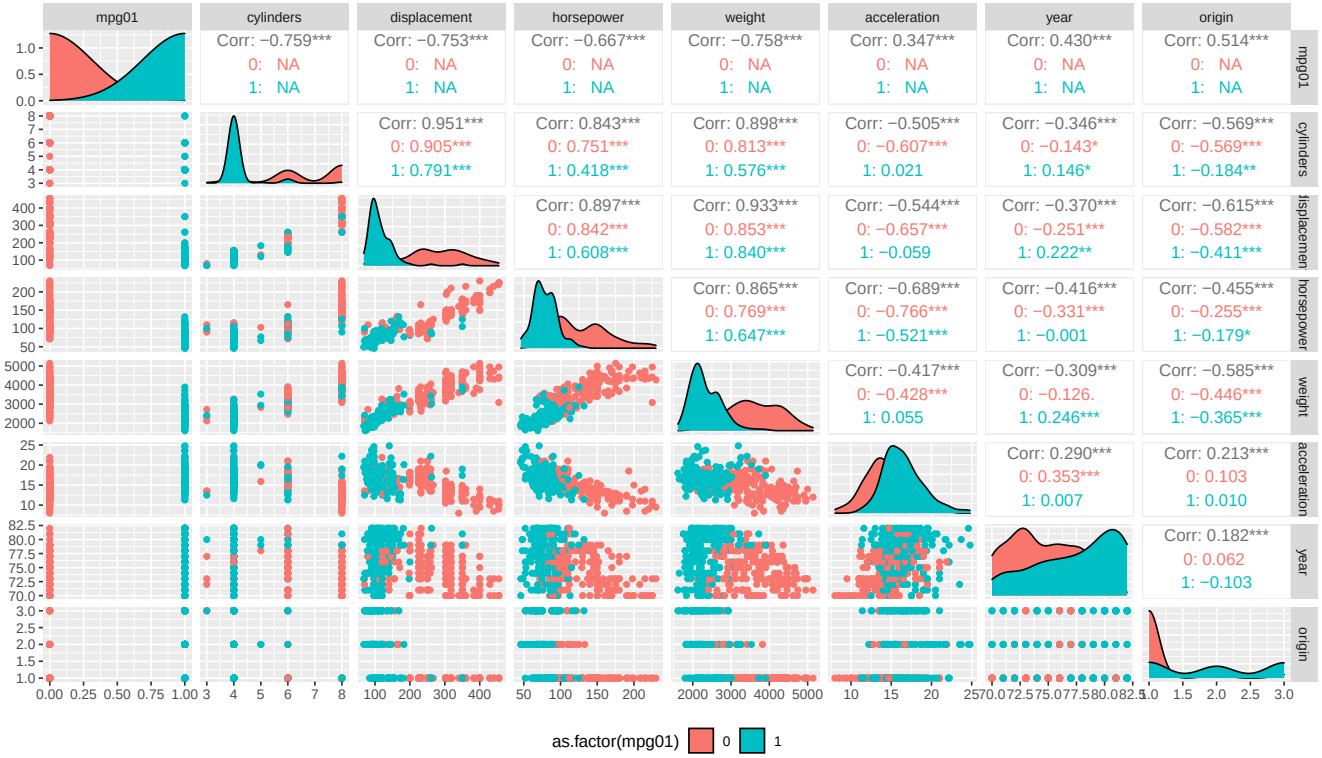


Figure 2: Pairs plot

We can clearly see from the pairs plot that cylinders, year, and origin will not be useful in predicting mpg01. The scatter plots show that displacement, horsepower, weight, and acceleration have possible influence on mpg01 and therefore can be used to predict the response variable. Thus, we will use displacement, horsepower, weight, and acceleration as the independent variables for this analysis.

Methodology

As previously mentioned, we will be building several classification models to predict the response variable in the testing set. To predict the binary variable **mpg01**, we will compare model results from Linear Discriminant Analysis(LDA), Quadratic Discriminant Analysis(QDA), Naive Bayes, Logistic Regression, K-nearest neighbors(KNN) with several K values, PCA-KNN with several K values, and Support Vector Machine.

We started the modeling by first creating 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. We then partitioned the data into training and testing sets using an random 70/30 split with replacement.

To fit Linear Discriminant Analysis model, **lda()** R function from the **MASS** package was used. For each observation in the training data, the response Y variable (i.e. mpg01) has a class label 0 or 1, and we have 4 independent X variables. The linear discriminant analysis or LDA algorithm, constructs 2 discriminant functions $d_2(x)$, that are linear functions of the 4 independent X variables and then classifies the data into the class that has a larger discriminant, or the $d_2(x)$ functions. To calculate the training and testing errors, we simply took the mean of classification errors.

To fit a Quadratic Discriminant Analysis model, **qda()** R function from MASS package was used. The QDA algorithm assumes that the data from classes 0 and 1 are normally distributed. Similar to lda, the training and testing errors were calculated as the mean of classification errors.

To fit a logistic regression model, **glm()** was used. To predict which class a data point belongs to, a probability cutoff of .5 was used because the data is somewhat balanced. So, if the prediction was greater or equal to **.5**, we assign it to class 1, otherwise we assign it to class 0. Similar to previous models, the training and testing errors were calculated as the mean of classification errors.

To fit a Naive Bayes model, **naive_bayed()** method from **naivebayes** library was used . Naive Bayes is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. Similar to previous models, the training and testing errors were calculated as the mean of classification errors

To fit a K-nearest neighbors model, **knn()** was used. For model simplicity, the following k values were used $K \in [1, 3, 5, 7, 9, 11, 13, 15]$. For each K value, a KNN model was built on a scaled training data, then tested on a scaled testing data. Similar to previous models, the training and testing errors were calculated as the mean of classification errors.

Additionally, two optional models were build (SVM and PCA-KNN). To fit Principal component K-nearest neighbors, we first needed to extract the top principal components from the data using principal component analysis. Principal component analysis is an integral part of dimensionality reduction in machine learning. The main idea is to find the linear combination of X variables that express as much of the variability in response variable as possible.

If the most variations of the predictors come out from the first few principal components, then there is enough to use them to build models.

Using the full data set, we extract the principal components and visualize the variance explained by each component (see below)

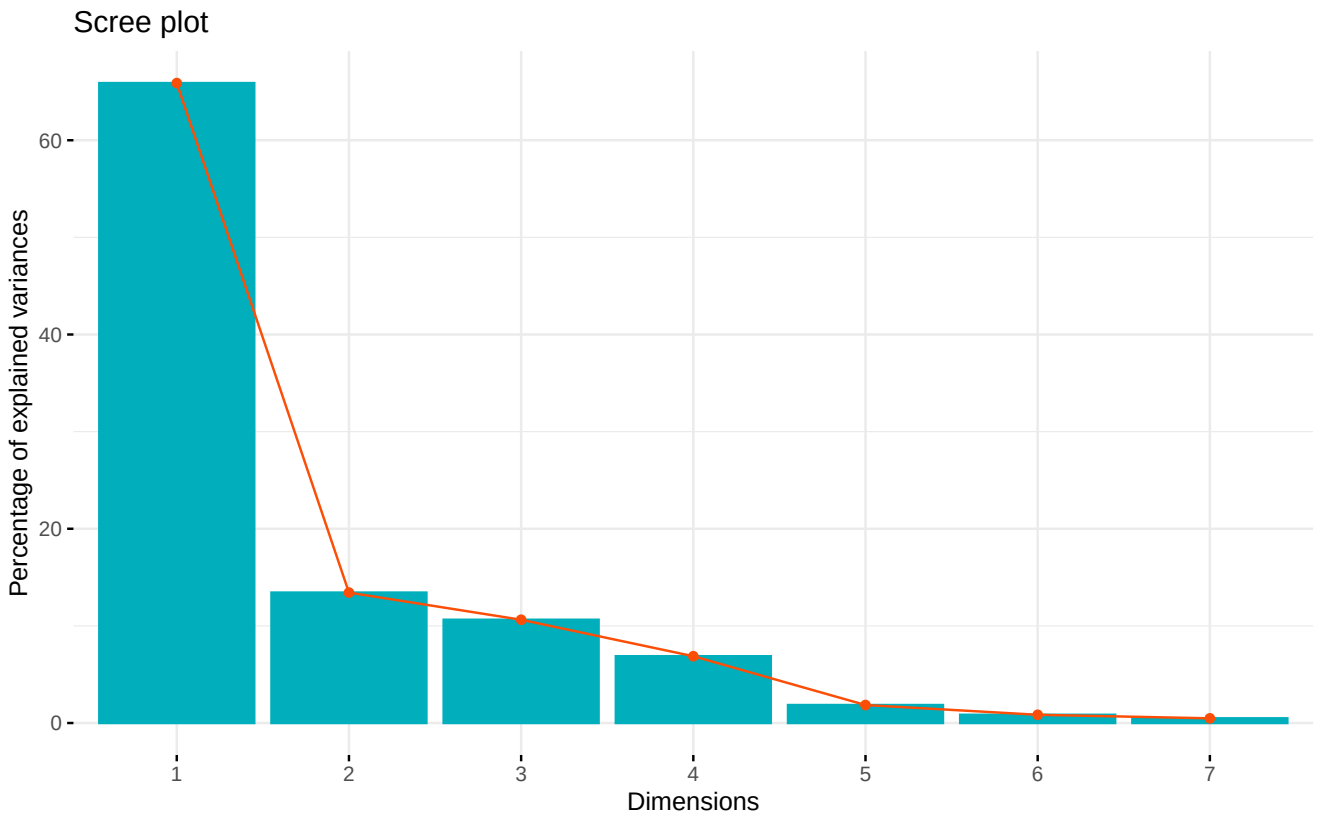


Figure 3: The explained variance of each principal component

We clearly notice from the scree plot that the first principal component has the highest explained variance (66%). We can also see that most of the variance is explained by the top four principal components. To see which variables are contributing the most to the explained variance, we can plot the PCA variables using **fviz_pca_var** R function (see below).

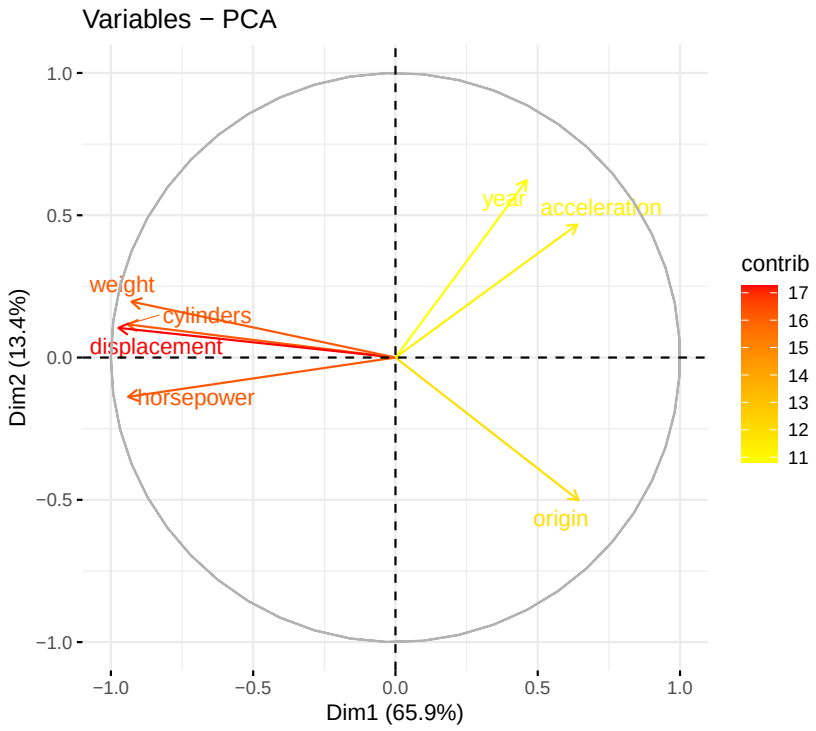


Figure 4: Variables that contributed the most to the explained variance

From the figure above, we can see the variables that contributed the most to the top four principal components. This confirms our initial choice of variable selection for other models.These four principal components were used to build a new dataset by combining them with the response variable. The dataset was then split using a random 70/30 split into training and testing. A new K-nearest neighbors model was fitted on the new training data and tested on the new testing data for the following K values: $K \in [1, 3, 5, 7, 9, 11, 13, 15]$.

For each K value, a new KNN model was built and the model performance was assessed. Similar to previous models, the training and testing errors were calculated as the mean of classification errors.

For SVM, **ksvm()** was used with **rbfdot** kernel and a tuning parameter $C = 100$. The model was fitted on a scaled data set. The objective of the support vector machine algorithm is to find a hyper-plane that distinctly classifies the data points. For the two classes case, The objective is to find a plane that has the maximum margin between the two classes. The training and testing errors for the model were calculated as the mean of the classification errors.

To decide which method produced the best results, Monte Carlo Cross-Validation was used to compare the average testing errors of each of the models. Monte Carlo Cross Validation algorithm splits the observed data points into training and testing subsets, and repeats the above computation for $B = 700$.

For the purpose of this homework, we are going to combine $n_1 = 276$ training data and $n_2 = 116$ testing data together into a larger data set. Then for each loop $b = 1, \ldots, B$ we will randomly select $n_1 = 276$ as a new training subset and use the remaining $n_2 = 116$ data as the new testing subset. Within each loop, we first build different

models from the training data of that specific loop and then evaluate their performances on the corresponding testing data. Therefore, for each model, we will obtain B values of the testing errors on B different subsets of testing data, denote by TE_b for $b = 1, \ldots, B$. Then **the average** performances of each model will be summarized by the sample mean and sample variances of these B values

Results

The training and testing errors for LDA, QDA, Logistic regression, Naive Bayes, and Support vector machine can be found below:

Table 3: Initial Summary of Training and Testing Errors

	LDA	QDA	Logit	NB	SVM
TrainingErrors	0.1107	0.0964	0.1143	0.1179	0.0071
TestingErrors	0.1250	0.1071	0.0982	0.0893	0.1161

The KNN training and testing errors are summarized in the table below:

Table 4: KNN Errors

K	Training.Error	Testing.Error
1	0.000000	0.107143
3	0.057143	0.098214
5	0.060714	0.089286
7	0.071429	0.098214
9	0.082143	0.116071
11	0.085714	0.107143
13	0.103571	0.107143
15	0.107143	0.107143

From the error table above, we clearly see that $K = 5$ has the lowest testing error compared to $K = 3$

The PCA-KNN training and testing errors are summarized in the table below:

Table 5: PCA-KNN Errors

K	Training.Error	Testing.Error
1	0.00000	0.08929
3	0.04643	0.08929
5	0.06786	0.08929
7	0.07500	0.10714
9	0.07500	0.11607
11	0.06786	0.10714
13	0.06429	0.09821
15	0.06786	0.10714

From the error table above, we clearly see that $K = 3$ has the lowest testing error and training errors.

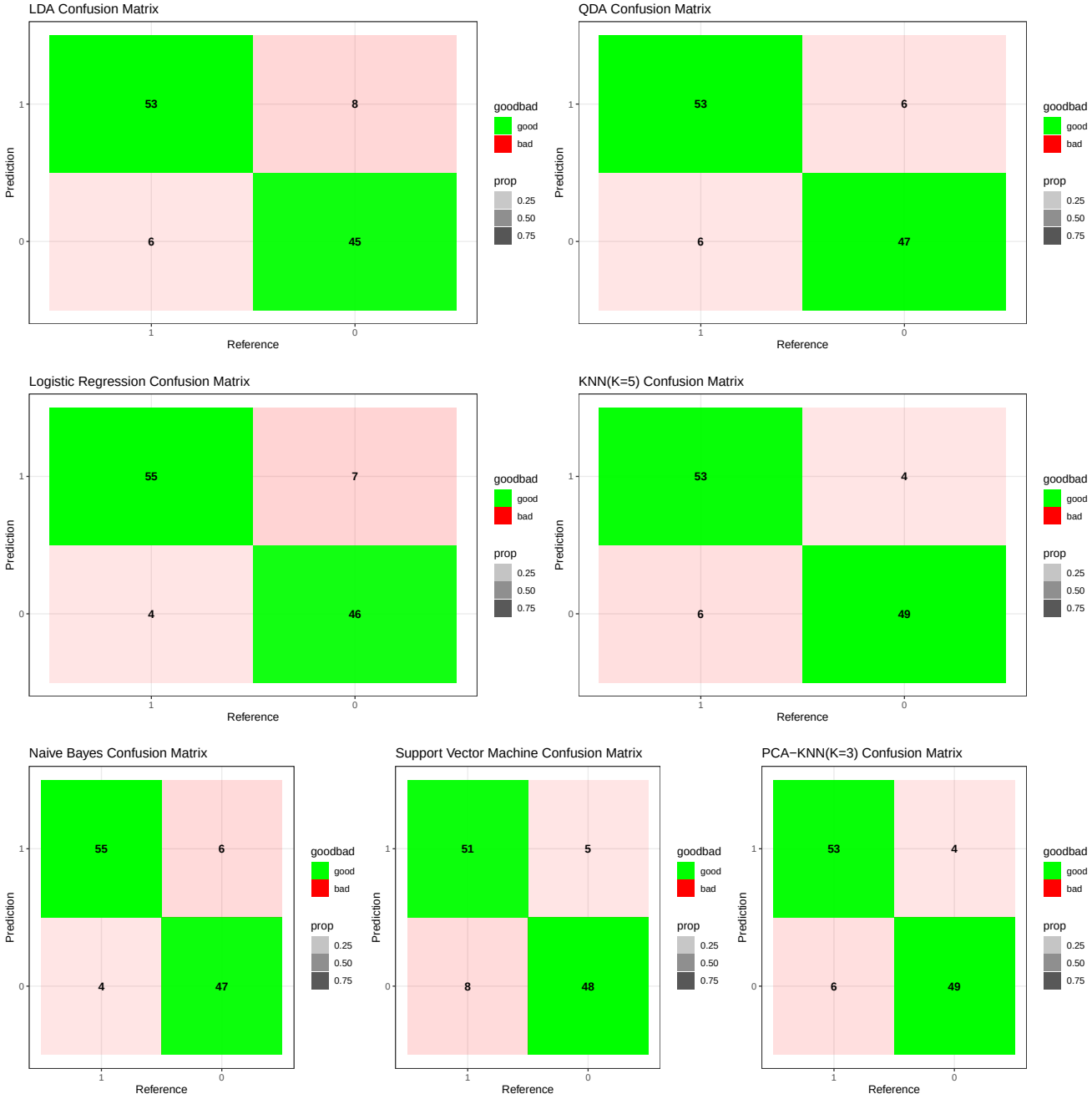
A full summary of the training and testing errors of all models used including KNN (K=5) and PCA-KNN (K=3) can be seen in the table below:

Table 6: Summary of Training and Testing Errors

	LDA	QDA	Logit	NB	SVM	KNN(K=5)	PCA-KNN(K=3)
TrainingErrors	0.1107	0.0964	0.1143	0.1179	0.0071	0.0607	0.0464
TestingErrors	0.1250	0.1071	0.0982	0.0893	0.1161	0.0893	0.0893

The models performed roughly the same. Naive Bayes, KNN (K=5), and PCA-KNN (K=3) performed the same on testing data. SVM model performed the best on training data, followed by PCA-KNN and KNN.

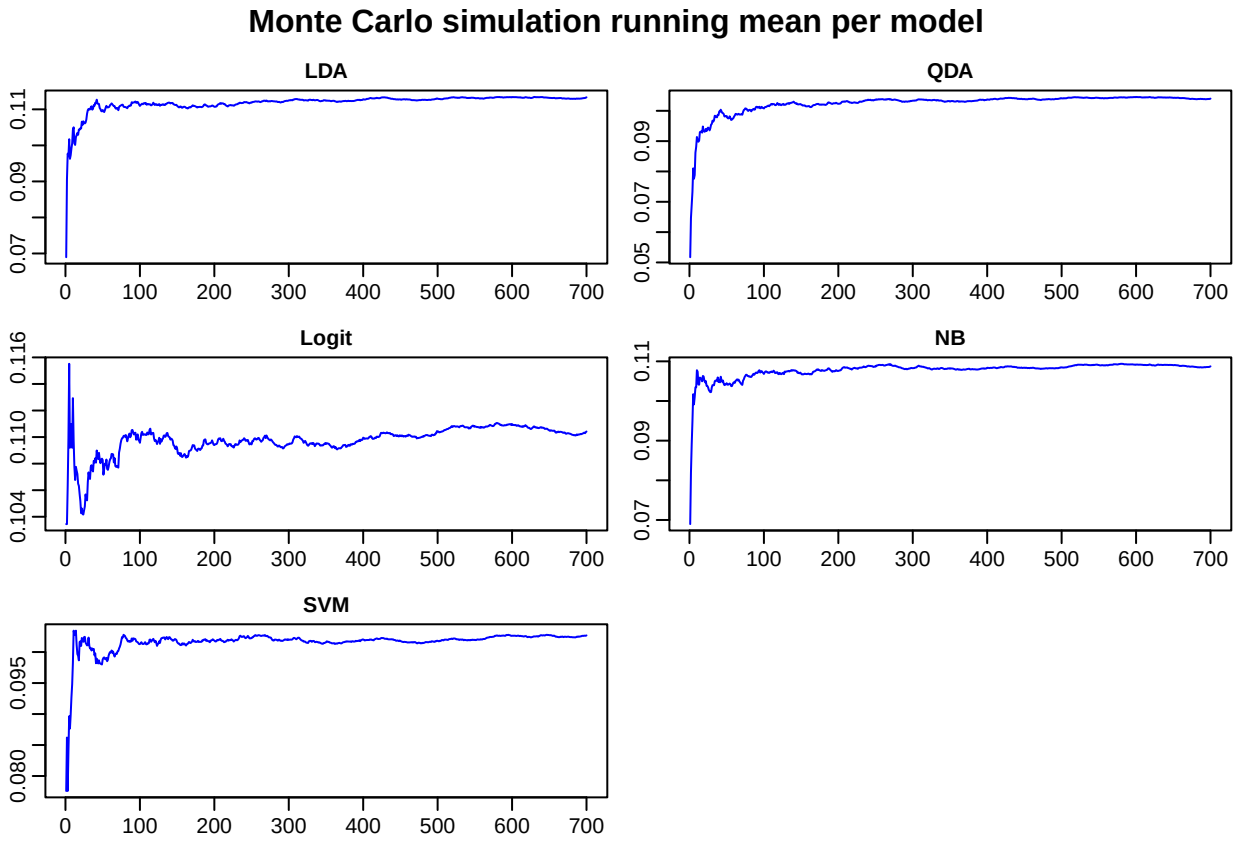
The results of each model's performance is summarized below in the following confusion matrices. The confusion matrices show how many records were misclassified by each model



The results from the Monte Carlo Cross-Validation on LDA, QDA, Logistic Regression, Naive Bayes, and SVM with $B = 700$ are as follows:

Table 7: Sample mean/variance of MCCV Errors for B = 700

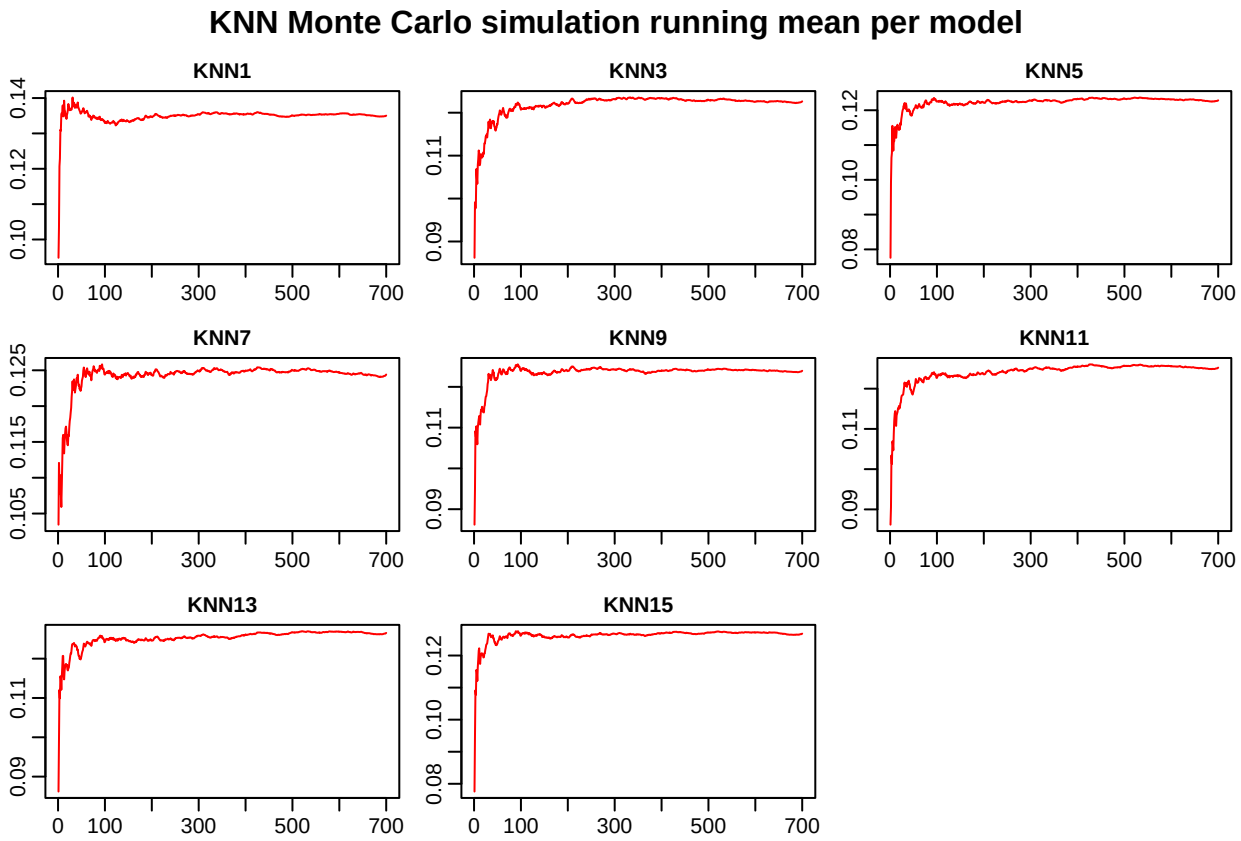
	LDA	QDA	Logit	NB	SVM
Mean	0.11334	0.10401	0.11042	0.10873	0.10268
Variance	0.00062	0.00058	0.00055	0.00056	0.00064



KNN Monte Carlo Cross Validation results:

Table 8: Sample mean/variance of MCCV Errors for B = 700

	KNN1	KNN3	KNN5	KNN7	KNN9	KNN11	KNN13	KNN15
Mean	0.13502	0.12264	0.12284	0.12441	0.12394	0.12523	0.12654	0.12682
Variance	0.00071	0.00068	0.00064	0.00066	0.00066	0.00070	0.00069	0.00067



After running Monte Carlo cross validation for $B = 700$ iteration, we see that SVM was the best performing model on test data followed by QDA then Naive Bayes.

Conclusion

In this homework, we explored different classification methods. Unlike the initial training/testing results, Monte Carlo cross validation models had an accuracy rate on testing data ranging from $\approx 87\%$ up to $\approx 89\%$. We also noticed that naive bayes model performed slightly better than logistic regression in cross validation. This might be because of the small sample size we used in both training and testing sets. According to academic research, it is common that naive bayes performs better than logistic regression even when the assumptions of naive bayes do not seem to hold.

Appendix

R Code

```
# Settings
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE, fig.pos = 'H')


# Check if packages are installed. If not, install them.
install.packages <- function(pkg) {
    new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
    if (length(new.pkg)) {
        install.packages(new.pkg)
    }
}


packages <- c("ggplot2", "mcmcplots", "knitr","xtable", "kableExtra","klaR","kernlab","kknn",
              "reshape2", "devtools","MASS","glmnet", "lares", "caret","GGally",
              "splines", "class", "dplyr", "PerformanceAnalytics",  "factoextra")


install.packages(packages)
library(mcmcplots)
library(knitr)
library(kableExtra)
library(ggplot2)
library(reshape2)
library(devtools)
library(lares)
library(caret)
library(splines)
library(class)
library(extrafont)
library(xtable)
library(dplyr)
library(leaps)
```

```
library(MASS)
library(lars)
library(pls)
library(glmnet)
library(tidyverse)
library(gtsummary)
library(tidyr)
library(PerformanceAnalytics)
library(corrplot)
library(klaR)
library(psych)
library(MASS)
library(GGally)
library(kernlab)
library(kknn)
library(class)
library(factoextra)
options(kableExtra.latex.load_packages = FALSE)
loadfonts()

devtools::install_github("haozhu233/kableExtra")
## Read data
Auto1 <- read.table(file = "Auto.csv", sep = ",", header=T);

dataTemp =  Auto1 %>% mutate(mpg01 = ifelse(Auto1$mpg >= median(Auto1$mpg), 1 , 0))%>%
            select(c("mpg01", "cylinders", "displacement","horsepower", "weight","acceleration", "year", "

numerical_data = dataTemp%>%
            select(c("displacement","horsepower", "weight","acceleration"))

knitr::kable(summary(dataTemp[1:4]),"pipe", digit=3,
             caption = paste("Summary statistics 1"))%>%
          kable_styling(position = "center", full_width = F)
```

```
knitr::kable(summary(dataTemp[5:8]),"pipe", digit=3,
             caption = paste("Summary statistics 2"))%>%
          kable_styling(position = "center", full_width = F)


ggplot(gather(dataTemp), aes(value)) +
    geom_histogram(bins = 35, fill = 'orange', color='black') +
    facet_wrap(~key, scales = 'free') +
    theme_classic()


ggpairs(dataTemp,legend = 1,
               mapping = ggplot2::aes(colour=as.factor(mpg01))) +
               theme(legend.position = "bottom")
# Set a seed
set.seed(42)
# Apply PCA
pca <- prcomp(dataTemp[,-1], scale = TRUE)
fviz_eig(pca, linecolor = "#FC4E07", barcolor = "#00AFBB", barfill = "#00AFBB")


# pca variables plot
fviz_pca_var(pca, col.var = "contrib", gradient.cols = c("yellow", "orange",
"red"),repel = TRUE)


set.seed(123)


# splitting the data
ind <- sample(2, nrow(dataTemp),
              replace = TRUE,
              prob = c(0.7, 0.3))


# Get training data
AutoTrain <- dataTemp[ind ==1,]%>%
            select(c("mpg01", "displacement","horsepower", "weight","acceleration"))


# get testing data
```

```
AutoTest <- dataTemp[ind ==2,]%>%
            select(c("mpg01", "displacement","horsepower", "weight","acceleration"))

training <- AutoTrain[,-1]

testing <- AutoTest[,-1]

ytrain <- AutoTrain$mpg01
ytest <- AutoTest$mpg01

TestingErrors  <- NULL;
TrainingErrors <- NULL;

### LDA
lda.model <- lda(mpg01~., AutoTrain)
pred <- predict(lda.model,newdata=testing)$class

pred1test <- predict(lda.model,newdata=testing)$class
pred1train <- predict(lda.model,newdata=training)$class

mod1train <- mean(pred1train != ytrain)
mod1test <- mean(pred1test != ytest)
TestingErrors <- c(TestingErrors, mod1test);
TrainingErrors <- c(TrainingErrors, mod1train);

lda.table <- data.frame(confusionMatrix(as.factor(pred) ,as.factor(ytest),
                                    dnn = c("Prediction", "Reference"))$table)

lda.plotTable <- lda.table %>%
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

lda.cm <- ggplot(data = lda.plotTable, mapping = aes(x = Reference,
```

```
                                    y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("LDA Confusion Matrix") +
  xlim(rev(levels(lda.plotTable$Reference)))



### QDA
qda.model = qda(mpg01~., data=AutoTrain)

qda.pred <- predict(qda.model,newdata=testing)$class

pred2test <- predict(qda.model,newdata=testing)$class
pred2train <- predict(qda.model,newdata=training)$class

mod2train <- mean(pred2train != ytrain)
mod2test <- mean(pred2test != ytest)
TestingErrors <- c(TestingErrors, mod2test);
TrainingErrors <- c(TrainingErrors, mod2train);

qda.table <- data.frame(confusionMatrix(as.factor(qda.pred) ,as.factor(ytest),
                                    dnn = c("Prediction", "Reference"))$table)



qda.plotTable <- qda.table %>%
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

qda.cm <- ggplot(data = qda.plotTable, mapping = aes(x = Reference,
                                    y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
```

```
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("QDA Confusion Matrix") +
  xlim(rev(levels(qda.plotTable$Reference)))

## Logistic Regression

glm <- glm(mpg01~., data=AutoTrain, family = binomial(link="logit"));

#calculate VIF values for each predictor variable in our model
# car::vif(glm)

glm.pred <- predict(glm, newdata=testing,type="response")

predicted.classes <- ifelse(glm.pred > 0.5, 1, 0)

# training and testing errors
pred3test <- predict(glm, newdata=testing,type="response")
pred3test <- ifelse(pred3test > 0.5, 1, 0)
pred3train <- predict(glm, newdata=training,type="response")
pred3train <- ifelse(pred3train > 0.5, 1, 0)

mod3train <- mean(pred3train != ytrain)
mod3test <- mean(pred3test != ytest)
TestingErrors <- c(TestingErrors, mod3test);
TrainingErrors <- c(TrainingErrors, mod3train);

glm.table <- data.frame(confusionMatrix(as.factor(predicted.classes) ,as.factor(ytest),
                                        dnn = c("Prediction", "Reference"))$table)

glm.plotTable <- glm.table %>%
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
  group_by(Reference) %>%
```

```
  mutate(prop = Freq/sum(Freq))

glm.cm <- ggplot(data = glm.plotTable, mapping = aes(x = Reference,
                                        y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("Logistic Regression Confusion Matrix") +
  xlim(rev(levels(glm.plotTable$Reference)))

# library(ROCR)       # For the ROC curve
# pr <- prediction(pred3test, ytest)
# prf <- performance(pr, measure = "tpr", x.measure = "fpr")
# plot(prf, colorize = TRUE, lwd=3)



## Naive Bayes
library(naivebayes)
bayes.model <- naive_bayes(as.factor(mpg01)~., data=AutoTrain, usekernel = T)

# plot(bayes.model)

bayes.pred <- predict(bayes.model,newdata=testing)



pred4test <- predict(bayes.model,newdata=testing)
pred4train <- predict(bayes.model,newdata=training)

mod4train <- mean(pred4train != ytrain)
mod4test <- mean(pred4test != ytest)
TestingErrors <- c(TestingErrors, mod4test);
TrainingErrors <- c(TrainingErrors, mod4train);
```

```
bayes.table <- data.frame(confusionMatrix(as.factor(bayes.pred) ,as.factor(ytest),
                                    dnn = c("Prediction", "Reference"))$table)


bayes.plotTable <- bayes.table %>%
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))


bayes.cm <- ggplot(data = bayes.plotTable, mapping = aes(x = Reference,
                                      y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("Naive Bayes Confusion Matrix") +
  xlim(rev(levels(bayes.plotTable$Reference)))


# SVM
set.seed(123) # a seed is recommended if u want reproducible results
svm <- ksvm(as.factor(mpg01)~., data=AutoTrain,type="C-svc",kernel="rbfdot",C=100,scaled=T)
svm.pred <- predict(svm, testing)


pred5test <- predict(svm,newdata=testing)
pred5train <- predict(svm,newdata=training)


mod5train <- mean(pred5train != ytrain)
mod5test <- mean(pred5test != ytest)
TestingErrors <- c(TestingErrors, mod5test);
TrainingErrors <- c(TrainingErrors, mod5train);


svm.table <- data.frame(confusionMatrix(as.factor(svm.pred) ,as.factor(ytest),
                                    dnn = c("Prediction", "Reference"))$table)


svm.plotTable <- svm.table %>%
```

```
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

svm.cm <- ggplot(data = svm.plotTable, mapping = aes(x = Reference,
                                      y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("Support Vector Machine Confusion Matrix") +
  xlim(rev(levels(svm.plotTable$Reference)))



error_summary <- rbind(TrainingErrors, TestingErrors)
colnames(error_summary) <- c( "LDA","QDA", "Logit", "NB", "SVM")

# displaying the tables
knitr::kable(error_summary,"pipe", digit=4,
             caption = paste("Initial Summary of Training and Testing Errors"))%>%
          kable_styling(position = "center", full_width = F)
res1train <- c()
testerror <- c()

k.vals = c(1,3,5,7, 9, 11, 13, 15);
# scale data for KNN
knn_train <- training %>%
             scale()%>%
             data.frame()

knn_test <- testing %>%
             scale()%>%
             data.frame()
```

```r
for (i in k.vals) {
  ypred2.train <- knn(knn_train, knn_train, ytrain, k=i);
  res1train <- c( res1train, mean( ypred2.train != ytrain));

}

for (i in k.vals) {
  ypred2.test <- knn(knn_train, knn_test, ytrain, k=i);
  testerror <- c(testerror, mean( ypred2.test != ytest));
  }

# plot
test_error_tbl <- data.frame(
  K = k.vals,
  Testing.Error = testerror
)

train_error_tbl <- data.frame(
  K = k.vals,
  Training.Error = res1train
)



error_data = cbind(train_error_tbl,test_error_tbl)
knitr::kable(error_data[, c(1, 2, 4)],"pipe", digit=6,caption = "KNN Errors")

# Set a seed
set.seed(42)
# Apply PCA
pca <- prcomp(dataTemp[,-1], scale = TRUE)
# pick the first four principal components
pcc <- pca$x[,1:4]

pcc_data <- cbind(mpg01 = dataTemp$mpg01, pcc) %>% as.data.frame()
```

```r
# train/test split
pcc_train <- pcc_data[ind ==1,]
pcc_test <- pcc_data[ind ==2,]

pcc_training <- pcc_train[,-1]

pcc_testing <- pcc_test[,-1]

pcc_ytrain <- pcc_train$mpg01
pcc_ytest <- pcc_test$mpg01

pcc_trainerror <- c()
pcc_testerror <- c()

for (i in k.vals) {
  pcc.ypred.train <- knn(pcc_training, pcc_training, pcc_ytrain, k=i);
  pcc_trainerror <- c( pcc_trainerror, mean( pcc.ypred.train != pcc_ytrain));

}

for (i in k.vals) {
  pcc.ypred.test <- knn(pcc_training, pcc_testing, pcc_ytrain, k=i);
  pcc_testerror <- c(pcc_testerror, mean( pcc.ypred.test != pcc_ytest));
  }

# plot
pcc_test_error_tbl <- data.frame(
  K = k.vals,
  Testing.Error = pcc_testerror
)

pcc_train_error_tbl <- data.frame(
  K = k.vals,
```

```
  Training.Error = pcc_trainerror
)



pcc_error_data = cbind(pcc_train_error_tbl,pcc_test_error_tbl)
knitr::kable(pcc_error_data[, c(1, 2, 4)],"pipe", digit=5,caption = "PCA-KNN Errors")


# building new models
pred6train <- knn(knn_train, knn_train, ytrain, k=5);
pred6test <- knn(knn_train, knn_test, ytrain, k=5);

mod6train <- mean(pred6train != ytrain)
mod6test <- mean(pred6test != ytest)
TestingErrors <- c(TestingErrors, mod6test);
TrainingErrors <- c(TrainingErrors, mod6train);

knn.table <- data.frame(confusionMatrix(as.factor(pred6test) ,as.factor(ytest),
                                        dnn = c("Prediction", "Reference"))$table)

knn.plotTable <- knn.table %>%
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

knn.cm <- ggplot(data = knn.plotTable, mapping = aes(x = Reference,
                                         y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("KNN(K=5) Confusion Matrix") +
  xlim(rev(levels(knn.plotTable$Reference)))

pred7train <- knn(pcc_training, pcc_training, pcc_ytrain, k=3);
```

```r
pred7test <- knn(pcc_training, pcc_testing, pcc_ytrain, k=3);

TestingErrors <- c(TestingErrors, mean(pred7test != pcc_ytest));
TrainingErrors <- c( TrainingErrors, mean(pred7train != pcc_ytrain));

pca.knn.table <- data.frame(confusionMatrix(as.factor(pred7test) ,as.factor(pcc_ytest),
                                    dnn = c("Prediction", "Reference"))$table)

pca.knn.plotTable <- pca.knn.table %>%
  mutate(goodbad = ifelse(Prediction == Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

pca.knn.cm <- ggplot(data = pca.knn.plotTable, mapping = aes(x = Reference,
                                      y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  ggtitle("PCA-KNN(K=3) Confusion Matrix") +
  xlim(rev(levels(pca.knn.plotTable$Reference)))

error_summary <- rbind(TrainingErrors, TestingErrors)
colnames(error_summary) <- c( "LDA","QDA", "Logit", "NB", "SVM", "KNN(K=5)", "PCA-KNN(K=3)")

# displaying the tables
knitr::kable(error_summary,"pipe", digit=4,
             caption = paste("Summary of Training and Testing Errors"))%>%
          kable_styling(position = "center", full_width = F)

lda.cm + qda.cm
glm.cm  + knn.cm
bayes.cm + svm.cm + pca.knn.cm
FullData <- rbind(AutoTrain, AutoTest)
```

```r
dim(AutoTrain)
dim(AutoTest)

### combine to a full data set
n1 = 276; # training set sample size

n = dim(FullData)[1]; ## the total sample size
set.seed(7406); ### set the seed for randomization

B = 700; ## number of loops
kvals <- c(1,3,5,7, 9, 11, 13, 15);
TEALL = NULL; ### Final TE values for testing errors
TEALL_KNN = NULL;
for (b in 1:B){
  ## randomly select n1 observations as new training subset in each loop
  flag <- sort(sample(1:n,n1));
  traintempset <- FullData[flag, ];
  testtempset <- FullData[-flag, ];

  test <- testtempset[,-1]
  ytrue <- testtempset[,1]

  #---------------------LDA-----------------------#

  lda.model <- lda(mpg01~., traintempset)
  pred1test <- predict(lda.model, newdata=test)$class
  te1 <- mean(pred1test != ytrue)

  #------------------ QDA -----------------------#
  qda.model = qda(mpg01~., data=traintempset)
  pred2test <- predict(qda.model,newdata=test)$class
  te2 <- mean(pred2test != ytrue)

  #--------------------Logistic Regression---------#
```

```
  glm <- glm(mpg01~., data=traintempset, family = binomial(link="logit"));
  pred3test <- predict(glm, newdata=test,type="response")
  pred3test <- ifelse(pred3test > 0.5, 1, 0)
  te3 <- mean(pred3test != ytrue)

  #-----------------------Naive Bayes-------------#
  bayes.model <- naive_bayes(as.factor(mpg01)~., data=traintempset, usekernel = T)
  pred4test <- predict(bayes.model,newdata=test)
  te4 <- mean(pred4test != ytrue)

  #-----------------------------SVM----------------------#
  svm <- ksvm(as.factor(mpg01)~., data=traintempset,type="C-svc",kernel="rbfdot",C=100,scaled=T)
  pred5test <- predict(svm,newdata=test)
  te5 <- mean(pred5test != ytrue)

  #-------------------------KNN---------------------------#
  temptesterror <- NULL;
  xnew2temp <- testtempset[,-1];
  for (i in 1: 8){
    kk <- kvals[i];
    ypred2temp <- knn(traintempset[,-1], xnew2temp, traintempset[,1], k=kk);
    temptesterror <- cbind(temptesterror, mean( ypred2temp != testtempset[,1]));
  }

  TEALL = rbind(TEALL, cbind(te1, te2, te3, te4, te5));
  TEALL_KNN = rbind(TEALL_KNN,temptesterror)
}

colnames(TEALL) <- c("LDA","QDA", "Logit", "NB", "SVM")

Mean = apply(TEALL, 2, mean);
Variance  = apply(TEALL, 2, var);

data = rbind(Mean, Variance)
```

```r
# displaying the tables
knitr::kable(data,"pipe", digit=5,
             caption = paste("Sample mean/variance of MCCV Errors for B = ", B))%>%
          kable_styling(position = "center", full_width = F)
par(cex.main=1)
rmeanplot(TEALL, style="plain", col="blue",
          plot.title = "Monte Carlo simulation running mean per model")
colnames(TEALL_KNN) <- c("KNN1", "KNN3", "KNN5",
                         "KNN7","KNN9", "KNN11", "KNN13", "KNN15")
Mean = apply(TEALL_KNN, 2, mean);
Variance  = apply(TEALL_KNN, 2, var);


data_knn = rbind(Mean, Variance)



# displaying the tables
knitr::kable(data_knn,"pipe", digit=5,
             caption = paste("Sample mean/variance of MCCV Errors for B = ", B))%>%
          kable_styling(position = "center", full_width = F)
par(cex.main=1)
rmeanplot(TEALL_KNN, style="plain", col="red",
          plot.title = "KNN Monte Carlo simulation running mean per model")
```