AMA565 ADVANCED HIGH DIMENSIONAL DATA ANALYSIS

Project on IOT Device Identification

Lui Man Hon, 20000963g

Au Hing Yu, 20041325g

Chan Yuet Fai Novena, 20071573g

2022/04/10

Introduction

'IOT device identification' is a dataset collected from Kaggle at https://www.kaggle.com/datasets/fanbyprinciple/iot-device-identification. The aim for this analysis is to classify IOT devices categories based on the web traffic. There are 298 columns in the dataset, including 297 parameters and 1 categorical output column. The training dataset contains 1000 observations while the testing dataset contains 900 observations.

```
library(rpart)
library(rpart.plot)
library(caret)
library(randomForest)
library(ggplot2)
library(reshape2)
library(factoextra)
library(MASS)
library(glmnet)
library(xgboost)
library(class)

train_data <- read.csv('/Users/novenachan/Downloads/iot_device/iot_devi
ce_train.csv',header=TRUE)
test_data <- read.csv('/Users/novenachan/Downloads/iot_device/iot_devic
e_test.csv',header=TRUE)
dim(train_data)

## [1] 1000  298

dim(test_data)

## [1] 900 298
```

Data Preparation

Validation on Missing Values

Upon checking, there is no missing value for both training and testing dataset.

```
is.null(train_data)

## [1] FALSE

is.null(test_data)

## [1] FALSE
```

Categorical Data Handling

The only character data is device_category. To ease analysis, device_category is converted to a number in alphabetical order. For example, 1 for baby_monitor, 5 for smoke_detector and 10 for water_sensor.

```
category <- sort(unique(train_data$device_category))
category

##  [1] "baby_monitor"    "lights"          "motion_sensor"   "security
_camera"
##  [5] "smoke_detector"  "socket"          "thermostat"      "TV"

##  [9] "watch"           "water_sensor"

train_data$device_category <- factor(train_data$device_category)
test_data$device_category <- factor(test_data$device_category)

for (i in c(1:length(levels(train_data$device_category)))){
  levels(train_data$device_category)[i] <- i}

for (i in c(1:length(levels(test_data$device_category)))){
  levels(test_data$device_category)[i] <- i}
```

Extract Input Parameters

For easy analysis, parameters are separated out from the dataset.

```
train_data_in <- train_data[-length(train_data)]
test_data_in <- test_data[-length(test_data)]
dim(train_data_in)

## [1] 1000  297

dim(test_data_in)

## [1] 900 297
```

Removal of Single Value Parameters

It is found that some parameters are having same value for all observations. As a result, the variance for these parameters equal to 0. This may cause error during calculation or analysis. Given the fact that same values would not affect the result or be helpful when doing classification, these dummy parameters are removed in advance.

There are 137 parameters with zero variance. After removal, 160 parameters are remained.

```
not_dummy_columns <- vapply(train_data_in, function(x) length(unique(x)
) > 1, logical(1L))
train_data_input <- train_data_in[not_dummy_columns]
test_data_input <- test_data_in[not_dummy_columns]
dim(train_data_input)

## [1] 1000  160

dim(test_data_input)

## [1] 900 160
```

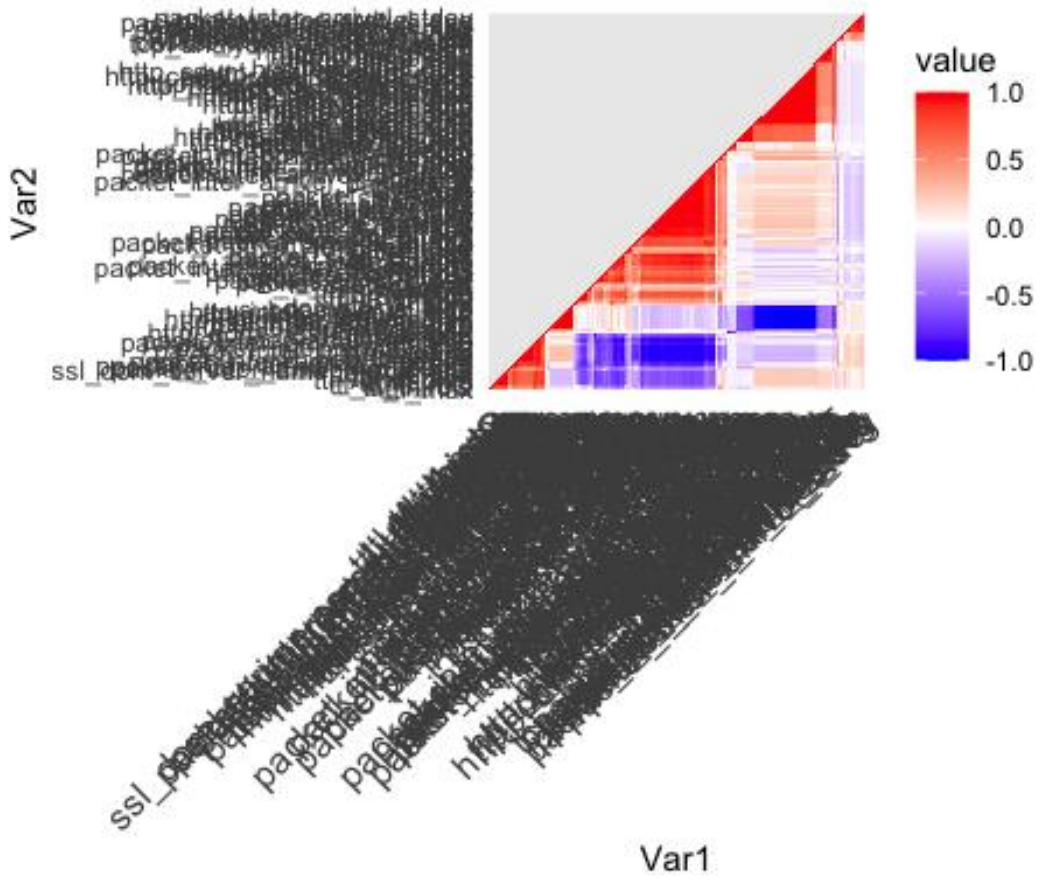
Data visualization

Correlation Heatmap

Looking at the correlation heatmap, it is clear that some parameters in the dataset are highly positive (in red) or negative (in blue) correlated.

```
get_lower_tri<-function(cormat){
    cormat[upper.tri(cormat)] <- NA
    return(cormat)
  }
reorder_cormat <- function(cormat){
     dd <- as.dist((1-cormat)/2)
    hc <- hclust(dd)
    cormat <-cormat[hc$order, hc$order]
}

cormat <- round(cor(train_data_input),2)
cormat <- reorder_cormat(cormat)
lower_tri <- get_lower_tri(cormat)
melted_cormat <- melt(lower_tri,na.rm=TRUE)

ggplot(data = melted_cormat, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile()+
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
   midpoint = 0, limit = c(-1,1), space = "Lab")+
```

```
  theme_minimal()+
  theme(axis.text.x = element_text(angle = 45, vjust = 1,
    size = 12, hjust = 1))+
  coord_fixed()
```



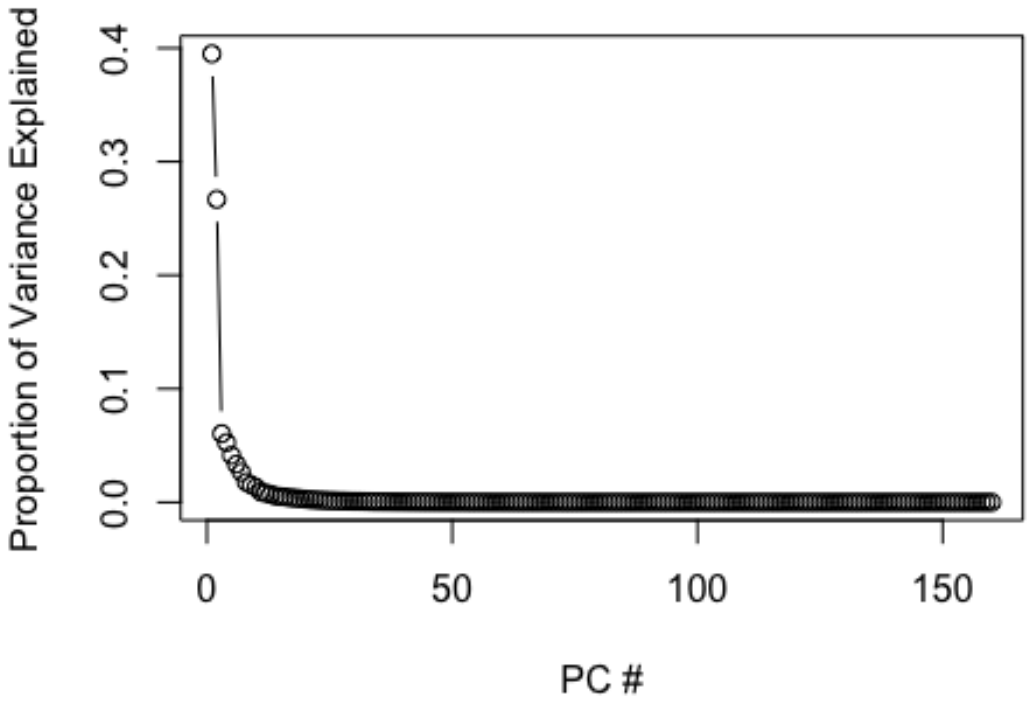
Principal Component Analysis

PCA is performed with scaling set to TRUE since some parameters are in different scale. For example, the parameter duration has a mean of 1.42 while packet_size_var has a mean of 18436.9.

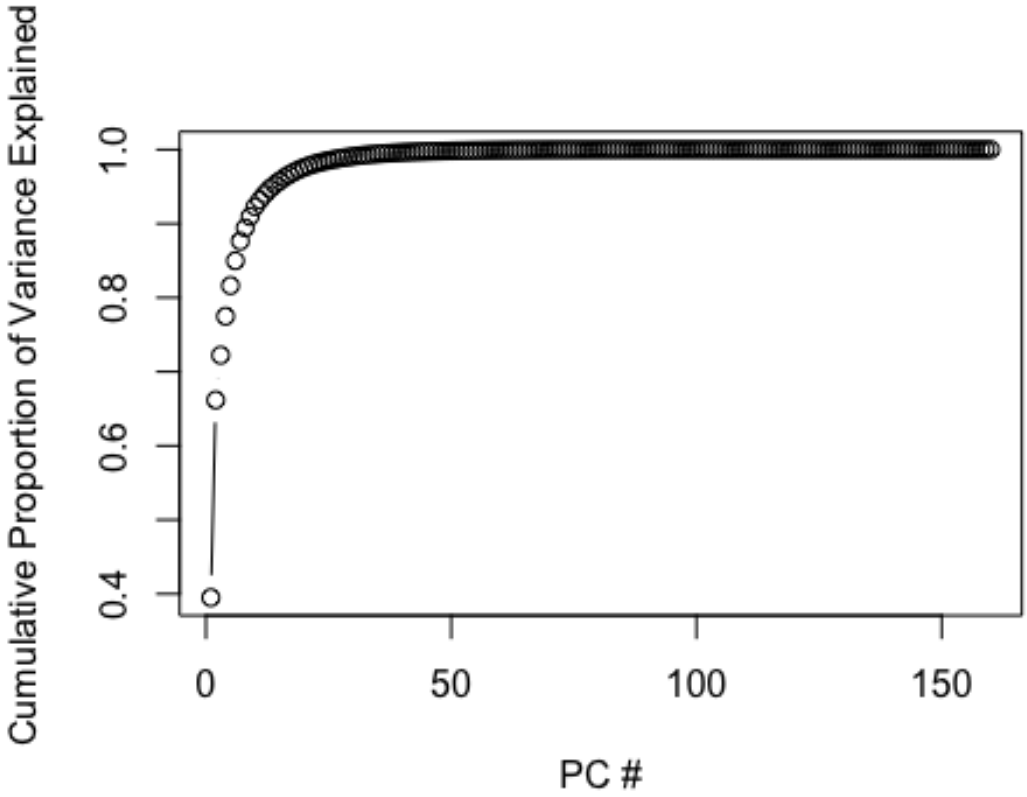
```
mean(train_data$duration)

## [1] 1.419692

mean(train_data$packet_size_var)

## [1] 18436.9

prin_comp <- prcomp(train_data_input, center=TRUE, scale = TRUE)
dim(prin_comp$x)

## [1] 1000  160
```

From the output result and plots of PCA, the variance can be explained by the first few PCs.

```
pr_var <- prin_comp$sdev^2
prop_var_ex <- pr_var/sum(pr_var)
plot(prop_var_ex, xlab = "PC #",
     ylab = "Proportion of Variance Explained",
     type = "b")
```

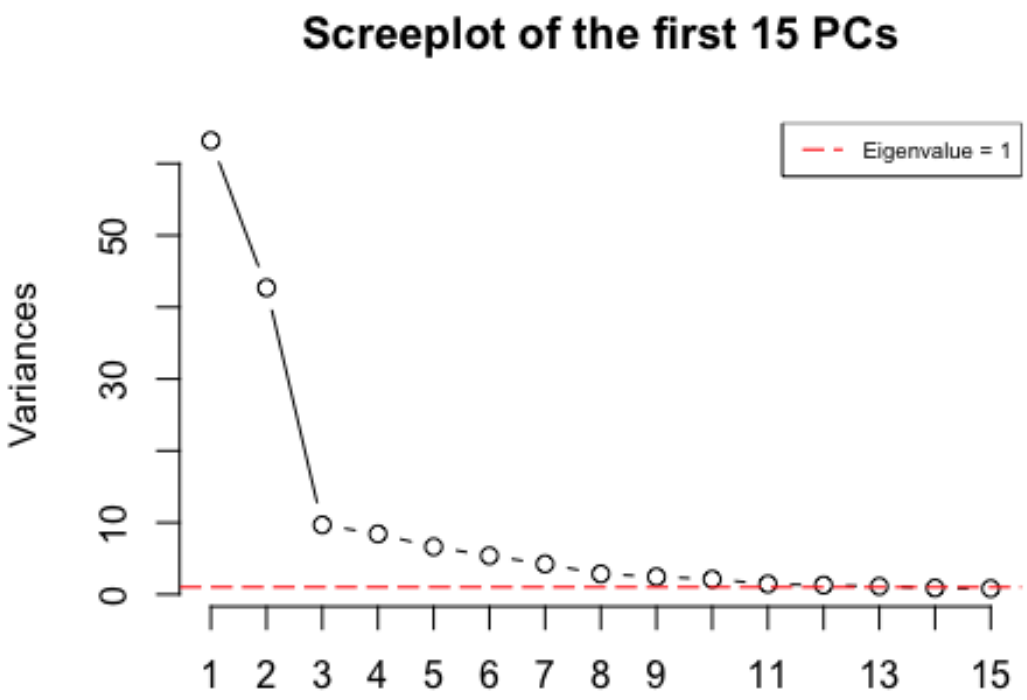



```
plot(cumsum(prop_var_ex), xlab = "PC #",
     ylab = "Cumulative Proportion of Variance Explained",
     type = "b")
```

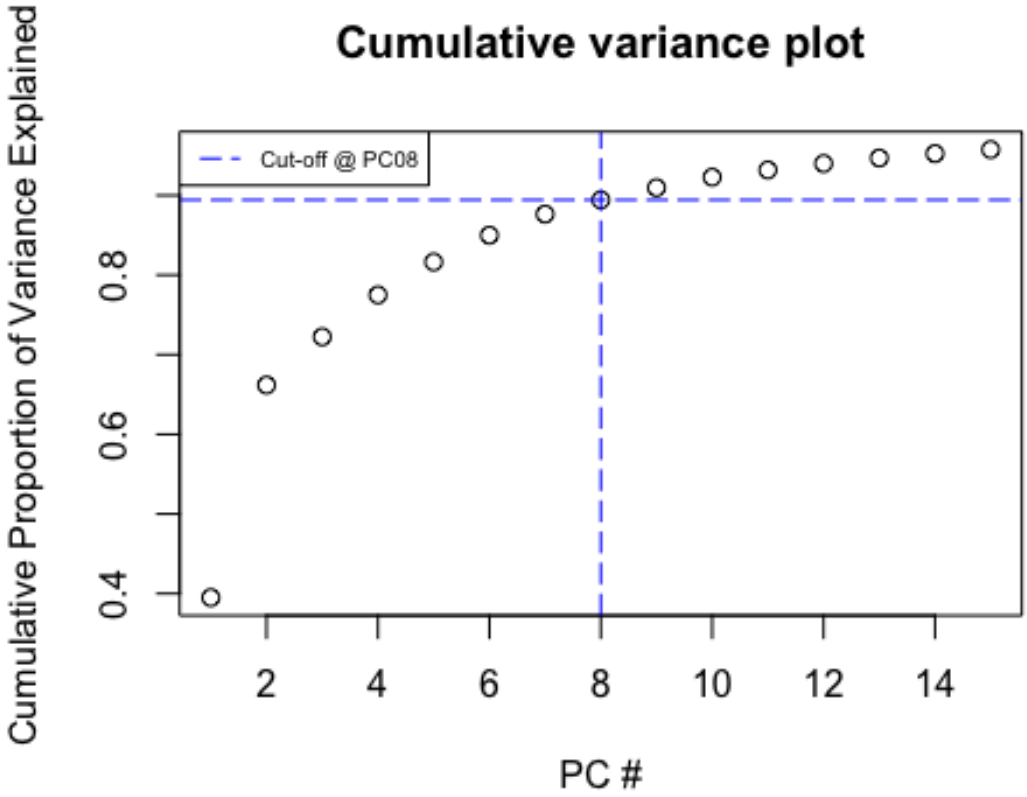


Looking closer at the first 15 PCs, it is found that the first 8 components have an eigenvalue > 1 which can explain almost 90% of variance. As a result, dimensions can be reduced from 160 to 8 with around 10% loosing of variance.

```
screeplot(prin_comp, type = "l", npcs = 15, main = "Screeplot of the fi
rst 15 PCs")
abline(h = 1, col="red", lty=5)
legend("topright", legend=c("Eigenvalue = 1"),
       col=c("red"), lty=5, cex=0.6)
```

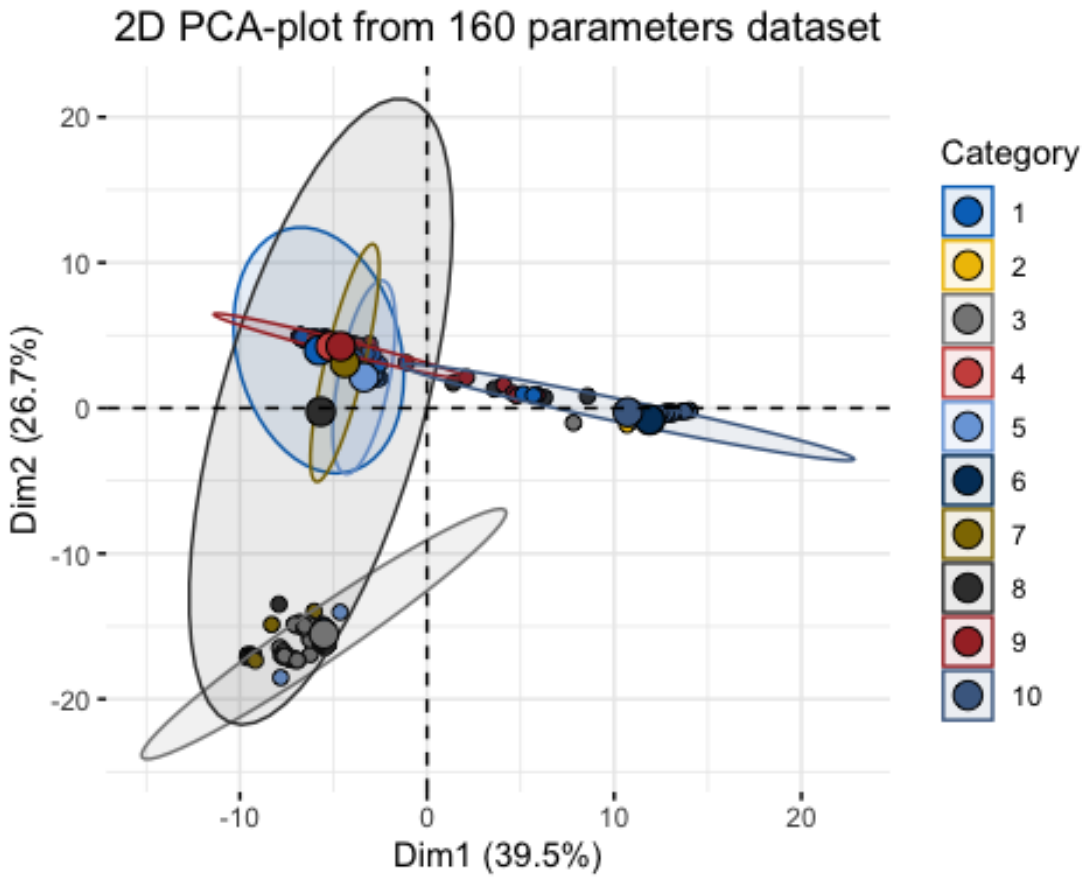



```
cumpro <- cumsum(prin_comp$sdev^2 / sum(prin_comp$sdev^2))
plot(cumpro[0:15], xlab = "PC #", ylab = "Cumulative Proportion of Vari
ance Explained", main = "Cumulative variance plot")
abline(v = 8, col="blue", lty=5)
abline(h = 0.89426, col="blue", lty=5)
legend("topleft", legend=c("Cut-off @ PC08"),
       col=c("blue"), lty=5, cex=0.6)
```



The first two components can explain more than 60% of variance. Although they may not be sufficient to classify 10 categories, it is clear that there is a separation of two groups.

```
fviz_pca_ind(prin_comp, geom.ind = "point", pointshape = 21,
             pointsize = 2,
             fill.ind = factor(train_data$device_category),
             col.ind = "black",
             palette = "jco",
             addEllipses = TRUE,
             label = "var",
             col.var = "black",
             repel = TRUE,
             legend.title = "Category") +
  ggtitle("2D PCA-plot from 160 parameters dataset") +
  theme(plot.title = element_text(hjust = 0.5))
```



Model Training

In this report, we use both 160 parameters and 8 principle components as the input to train 7 different models. For each input, we first validate the model with the training dataset itself, then perform prediction using the testing dataset.

Training and testing data for 160 parameters with standardization

```
train.all <- data.frame(device_category = train_data$device_category, t
rain_data_input)
test.all <- data.frame(device_category = test_data$device_category, tes
t_data_input)

colmean <- apply(train.all[2:length(train.all)],2,mean)
colsd <- apply(train.all[2:length(train.all)],2,sd)

#Standardization for training dataset
for (i in c(2:length(train.all))){
   train.all[i]<- (train.all[i]-colmean[i-1])/colsd[i-1]
}

#Standardization for testing dataset
```

```
for (i in c(2:length(test.all))){
   test.all[i]<- (test.all[i]-colmean[i-1])/colsd[i-1]
}

cat("The dimensions of train.all is", dim(train.all))

## The dimensions of train.all is 1000 161

cat("\nThe dimensions of test.all is", dim(test.all))

##
## The dimensions of test.all is 900 161
```

Training and testing data for 8 principle components (8 PCs)

```
num_of_pc = 8
train.pcs <- data.frame(device_category = train_data$device_category, p
rin_comp$x)
train.8pc <- train.pcs[,0:num_of_pc+1]

test.pcs <- predict(prin_comp, newdata = test_data_input)
test.pcs <- data.frame(device_category = test_data$device_category, tes
t.pcs)
test.pcs <- as.data.frame(test.pcs)
test.8pc <- test.pcs[,0:num_of_pc+1]

cat("The dimensions of train.data.all is", dim(train.8pc))

## The dimensions of train.data.all is 1000 9

cat("\nThe dimensions of test.data.all is", dim(test.8pc))

##
## The dimensions of test.data.all is 900 9
```

The output result are stored into array of 'Accuracy', 'Model' and 'Type' which will be used as summary at the end of this report.

Accuracy refers to the prediction accuracy. Model refers to the training model that is applied. Type refers to either 'Train - 160', 'Train - 8pc', 'Test - 160', or 'Test - 8pc', corresponding to validation of training set with 160 parameters, validation of training set with 8 PCs, predication of testing set with 160 parameters and prediction of testing set with 8 PCs.

```
Accuracy <- c()
Model <- c()
Type <- c()
```

1. Decision Tree Classifer

With 160 parameters

Training a Decision Tree classifier

```
model <- rpart(device_category ~ .,data = train.all, method = "class")
```

Validation on training dataset

```
predicted <- predict(model, train.all, type="class")
accuracy <- mean(as.character(predicted) == as.character(train_data$dev
ice_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Decision Tree")
Type <- c(Type, "Train - 160")
```

Validation accuracy of training dataset

```
cat("The validation accuracy of Decision Tree Classifier with 160 param
eters is:", accuracy, "\n")

## The validation accuracy of Decision Tree Classifier with 160 paramet
ers is: 0.812

as.matrix(table(Actual = train_data$device_category, Predicted = predic
ted))

##       Predicted
## Actual  1  2  3  4  5  6  7  8  9 10
##     1  88  0  0  1  0  0  0 11  0  0
##     2   0 78  0  0  0 14  0  0  0  8
##     3   0  3 96  0  0  0  0  0  0  1
##     4   0  0  0 99  0  0  0  0  1  0
##     5   0  0  0  0 99  0  0  0  0  1
##     6   0 48  0  0  0 25  0  0  0 27
##     7   0  0  0  0  0  0 90 10  0  0
##     8   0  0  0  4  0  0  2 93  1  0
##     9   0  0  0  1  0  0  2 12 85  0
##     10  0 37  0  0  0  4  0  0  0 59
```

Prediction on testing dataset

```
predicted <- predict(model, test.all, type="class")
accuracy <- mean(as.character(predicted) == as.character(test_data$devi
ce_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Decision Tree")
Type <- c(Type, "Test - 160")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy of Decision Tree Classifier with 160 param
eters is:", accuracy, "\n")
```

```
## The prediction accuracy of Decision Tree Classifier with 160 paramet
ers is: 0.5277778
```

```
as.matrix(table(Actual = test_data$device_category, Predicted = predict
ed))
```

```
##       Predicted
## Actual   1   2   3   4   5   6   7   8   9  10
##      1  98   0   0   0   0   0   2   0   0   0
##      2   0  53   0   0   0  20   0   0   0  27
##      3   3   0  43   0  20   0   1  14  19   0
##      4  34   0   0  50   0   0   0  16   0   0
##      5   0   0   0   0 100   0   0   0   0   0
##      6   0  31   0   0   0   5   0   0   0  64
##      7   0   0   0  20   0   0   0  80   0   0
##      8   0   0   0   2   0   0   1  81  16   0
##      9   0   0   0   0   0   0   6  49  45   0
```

With 8 PCs

Training a Decision Tree Classifier

```
model <- rpart(device_category ~ .,data = train.8pc, method = "class")
```

Validation on training dataset

```
predicted <- predict(model, train.8pc, type="class")
accuracy <- mean(as.character(predicted) == as.character(train_data$dev
ice_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Decision Tree")
Type <- c(Type, "Train - 8 PCs")
```

Validation accuracy of training dataset

```
cat("The validation accuracy of Decision Tree Classifier with 8 PCs is:
", accuracy, "\n")
```

```
## The validation accuracy of Decision Tree Classifier with 8 PCs is: 0
.768
```

```
as.matrix(table(Actual = train_data$device_category, Predicted = predic
ted))
```

```
##       Predicted
## Actual  1  2  3  4  5  6  7  8  9 10
##     1  79  0  0  1  4  0  0 12  4  0
```

```
##      2   0 79  0  0  0 21  0  0  0  0
##      3   0  1 93  0  3  3  0  0  0  0
##      4   2  0  0 96  0  0  0  1  1  0
##      5   1  0  0  0 95  0  3  1  0  0
##      6   0 52  0  0  0 47  0  0  0  1
##      7   1  0  0  9  0  0 85  4  1  0
##      8   5  0  0  1  2  0  3 78 11  0
##      9   5  0  0  0  2  0  3  3 87  0
##      10  8 24  0  0  0 39  0  0  0 29
```

Prediction on testing dataset

```
predicted <- predict(model, test.8pc, type="class")
accuracy <- mean(as.character(predicted) == as.character(test_data$devi
ce_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Decision Tree")
Type <- c(Type, "Test - 8 PCs")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy of Decision Tree Classifier with 8 PCs is:
", accuracy, "\n")
```

```
## The prediction accuracy of Decision Tree Classifier with 8 PCs is: 0
.2311111
```

```
as.matrix(table(Actual = test_data$device_category, Predicted = predict
ed))
```

```
##       Predicted
## Actual  1  2  3  4  5  6  7  8  9 10
##      1  0  0  0  0 14  0  0 74 12  0
##      2  0  0  0  0  0 78  0  0  0 22
##      3  0  0 58  0 34  0  0  8  0  0
##      4  0  0 34  0 48  0  0 18  0  0
##      5  0  0  0  0 77  0  0  0 23  0
##      6  0  0  0  0  0 57  0  0  0 43
##      7  0  0  0  0 99  0  0  1  0  0
##      8  0  0  0  0 82  0  0 11  7  0
##      9  0  0  0  0 90  0  1  4  5  0
```

Conclusion for Decision Tree Classifier

For Decision Tree Classifier, training with 160 parameters has a better result than 8 PCs.

2. Multi-Class Linear Discriminant Analysis (LDA)

With 160 parameters

Training a LDA model

```
f_lda_all <- paste(names(train.all[1]),"~", paste(names(train.all)[-1],
collapse=" + "))
model <- lda(as.formula(paste(f_lda_all)), data=train.all)
```

Validation on training dataset

```
predicted <- predict(model, train.all)
accuracy <- mean(as.character(predicted$class) == as.character(train_da
ta$device_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "LDA")
Type <- c(Type, "Train - 160")
```

Validation accuracy of training dataset

```
cat("The validation accuracy of LDA with 160 parameters is:", accuracy,
 "\n")

## The validation accuracy of LDA with 160 parameters is: 0.852

as.matrix(table(Actual = train_data$device_category, Predicted = predic
ted$class))

##        Predicted
## Actual   1   2   3   4   5   6   7   8   9  10
##      1  100   0   0   0   0   0   0   0   0   0
##      2    0  87   0   0   0  13   0   0   0   0
##      3    0   1  96   0   0   3   0   0   0   0
##      4    0   0   0 100   0   0   0   0   0   0
##      5    0   0   0   0  99   0   0   0   0   1
##      6    0  58   0   0   0  38   0   0   0   4
##      7    0   0   0   0   0   0 100   0   0   0
##      8    0   0   0   0   0   0   0  99   1   0
##      9    0   0   0   0   0   0   1   3  96   0
##      10   0  37   0   0   0  26   0   0   0  37
```

Prediction on testing dataset

```
predicted <- predict(model, test.all)
accuracy <- mean(as.character(predicted$class) == as.character(test_dat
a$device_category))

Accuracy <- c(Accuracy, accuracy)
```

```
Model <- c(Model, "LDA")
Type <- c(Type, "Test - 160")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with LDA and 160 parameters is:", accuracy
, "\n")

## The prediction accuracy with LDA and 160 parameters is: 0.1488889

as.matrix(table(Actual = test_data$device_category, Predicted = predict
ed$class))

##       Predicted
## Actual  1  2  3  4  5  6  7  8  9 10
##      1  0  0  0 74 12  0  0 13  0  1
##      2  0 81  0  0  0  0  0  0  0 19
##      3 12  0  3 21  0  0  0 29 34  1
##      4  0  0  0  0  8  0 15 67  0 10
##      5 28  0 23  0 49  0  0  0  0  0
##      6  0 55  0  0  0  1  0  0  0 44
##      7  0  0  0 99  1  0  0  0  0  0
##      8 79  0  0  5  8  8  0  0  0  0
##      9 92  0  0  2  3  2  0  0  0  1
```

With 8 PCs

Training a LDA model

```
f_lda_8pc <- paste(names(train.8pc[1]),"~", paste(names(train.8pc)[-1],
collapse=" + "))
model <- lda(as.formula(paste(f_lda_8pc)), data=train.8pc)
```

Validation on training dataset

```
predicted <- predict(model, train.8pc)
accuracy <- mean(as.character(predicted$class) == as.character(train_da
ta$device_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "LDA")
Type <- c(Type, "Train - 8 PCs")
```

Validation accuracy for training dataset

```
cat("The validation accuracy with LDA and 8 PCs is:", accuracy, "\n")

## The validation accuracy with LDA and 8 PCs is: 0.617

as.matrix(table(Actual = train_data$device_category, Predicted = predic
ted$class))
```

```
##        Predicted
## Actual  1  2  3  4  5  6  7  8  9 10
##      1  56  0  0 11  0  0  0  0 33  0
##      2   0 98  0  0  0  2  0  0  0  0
##      3   0  5 95  0  0  0  0  0  0  0
##      4  48  0  0  0  0  0 12 37  3  0
##      5   2  0  0  0 98  0  0  0  0  0
##      6   0 87  0  0  0  6  0  0  0  7
##      7   0  0  0  0  0  0 89  3  8  0
##      8   6  0  0  4  0  0 10 77  3  0
##      9   7  0  0  6  0  0  7 13 67  0
##      10  8 60  0  0  0  1  0  0  0 31
```

Prediction on testing dataset

```
predicted <- predict(model, test.8pc)
accuracy <- mean(as.character(predicted$class) == as.character(test_dat
a$device_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "LDA")
Type <- c(Type, "Test - 8 PCs")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with LDA and 8 PCs is:", accuracy, "\n")

## The prediction accuracy with LDA and 8 PCs is: 0.3388889

as.matrix(table(Actual = test_data$device_category, Predicted = predict
ed$class))

##        Predicted
## Actual  1  2  3  4  5  6  7  8  9 10
##      1 55  0  0  0  1  0 13  0 31  0
##      2  0 78  0  0  0  0  0  0  0 22
##      3  2  0 89  0  5  0  1  3  0  0
##      4  2  0 50  0  0  0 28  4 16  0
##      5  0  0 50  0 50  0  0  0  0  0
##      6  0 49  0  0  0  7  0  0  0 44
##      7  1  0 87  0  0  0 12  0  0  0
##      8  1  0 79  2  0  0  1  9  8  0
##      9  0  0 89  2  0  0  2  2  5  0
```

Conclusion for Multi-Class LDA

For Multi-Class LDA, training with 8 PCs has a better result than 160 parameters.

3. Logistic Regression Classifier

With 160 parameters

Training a Logistic Regression Classifier

In the logistic regression model, MaxNWts has been used to control the maximium number of weights to 2000, as error 'Too many (1620) weights) has been returned without setting the maximium number. This fix is referencing a Stackoverflow post in 2017 (submartingale, 2017).

```
model <- nnet::multinom(device_category ~ .,data = train.all, MaxNWts=2
000)

## # weights:  1620 (1449 variable)
## initial  value 2302.585093
## iter  10 value 596.778100
## iter  20 value 457.000396
## iter  30 value 389.445434
## iter  40 value 349.762457
## iter  50 value 313.903490
## iter  60 value 298.830726
## iter  70 value 289.027345
## iter  80 value 286.539486
## iter  90 value 285.861257
## iter 100 value 285.523901
## final  value 285.523901
## stopped after 100 iterations
```

Validation on training dataset

```
predicted <- predict(model, train.all)
accuracy <- mean(as.character(predicted) == as.character(train.all$devi
ce_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Logistic")
Type <- c(Type, "Train - 160")
```

Validation accuracy for training dataset

```
cat("The validation accuracy with Logistic Regression Classifier and 16
0 parameters is:", accuracy, "\n")

## The validation accuracy with Logistic Regression Classifier and 160
parameters is: 0.859

as.matrix(table(Actual = train.all$device_category, Predicted = predict
ed))
```

```
##        Predicted
## Actual   1   2   3   4   5   6   7   8   9  10
##      1  100   0   0   0   0   0   0   0   0   0
##      2    0  84   0   0   0  11   0   0   0   5
##      3    0   1  96   0   0   2   0   0   0   1
##      4    0   0   0 100   0   0   0   0   0   0
##      5    0   0   0   0 100   0   0   0   0   0
##      6    0  57   0   0   0  35   0   0   0   8
##      7    0   0   0   0   0   0 100   0   0   0
##      8    0   0   0   0   0   0   0 100   0   0
##      9    0   0   0   0   0   0   0   0 100   0
##      10   0  33   0   0   0  23   0   0   0  44
```

Prediction on testing dataset

```
predicted <- predict(model, test.all)
accuracy <- mean(as.character(predicted) == as.character(test.all$devic
e_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Logistic")
Type <- c(Type, "Test - 160")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with Logistic Regression Classifier and 16
0 parameters is:", accuracy, "\n")

## The prediction accuracy with Logistic Regression Classifier and 160
parameters is: 0.07444444

as.matrix(table(Actual = test.all$device_category, Predicted = predicte
d))

##        Predicted
## Actual  1  2  3  4  5  6  7  8  9 10
##      1  1  0  0 73 13  0  0 13  0  0
##      2  0  1  0  0  0 81  1  0  0 17
##      3 16  0  1 19  1  0  2 59  2  0
##      4  0  0 32  0 16  2  0 18 32  0
##      5 77  0  0  0  0  0  0  0 23  0
##      6  0  2  0  0  0 58  0  0  0 40
##      7  0  0  0  0  1  0  0 99  0  0
##      8 77  0  0  0  8  0  8  6  0  1
##      9 89  0  0  1  3  0  3  4  0  0
```

With 8 PCs

Training a Logistic Regression Classifier

```
model <- nnet::multinom(device_category ~ .,data = train.8pc)
```

```
## # weights:  100 (81 variable)
## initial  value 2302.585093
## iter  10 value 898.867837
## iter  20 value 831.216186
## iter  30 value 768.203138
## iter  40 value 718.015159
## iter  50 value 677.453592
## iter  60 value 627.234820
## iter  70 value 565.836243
## iter  80 value 535.996224
## iter  90 value 512.780122
## iter 100 value 499.101502
## final  value 499.101502
## stopped after 100 iterations
```

Validation on training dataset

```
predicted <- predict(model, train.8pc)
accuracy <- mean(as.character(predicted) == as.character(train.8pc$devi
ce_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Logistic")
Type <- c(Type, "Train - 8 PCs")
```

Validation accuracy for training dataset

```
cat("The validation accuracy with Logistic Regression Classifier and 8
PCs is:", accuracy,"\n")

## The validation accuracy with Logistic Regression Classifier and 8 PC
s is: 0.799

as.matrix(table(Actual = train.8pc$device_category, Predicted = predict
ed))

##       Predicted
## Actual  1  2  3  4  5  6  7  8  9 10
##     1  93  0  0  2  0  0  0  3  2  0
##     2   0 94  0  0  0  6  0  0  0  0
##     3   0  3 96  0  0  1  0  0  0  0
##     4   1  0  0 94  0  0  3  0  2  0
##     5   0  0  0  0 99  0  0  0  0  1
##     6   0 77  0  0  0 18  0  0  0  5
##     7   1  0  0  3  0  0 90  5  1  0
##     8   5  0  0  4  0  0  3 85  3  0
##     9   3  0  0  1  0  0  2  2 92  0
##     10  0 48  0  0  0 14  0  0  0 38
```

Prediction on testing dataset

```
predicted <- predict(model, test.8pc)
accuracy <- mean(as.character(predicted) == as.character(test_data$devi
ce_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Logistic")
Type <- c(Type, "Test - 8 PCs")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with Logistic Regression Classifier and 8
PCs is:", accuracy,"\n")

## The prediction accuracy with Logistic Regression Classifier and 8 PC
s is: 0.2566667

as.matrix(table(Actual = test_data$device_category, Predicted = predict
ed))

##        Predicted
## Actual   1   2   3   4   5   6   7   8   9  10
##      1   0   0   0  87   0   0   0  13   0   0
##      2   0  63   0   0   0  15   0   0   3  19
##      3  34   0   2   2  27   0  33   2   0   0
##      4   0   0   0  34   0   0  34  32   0   0
##      5   0   0   0   0  99   0   1   0   0   0
##      6   0  31   0   0   0  31   0   0   1  37
##      7   0   0   0 100   0   0   0   0   0   0
##      8   2   0   0  21  41   0  35   1   0   0
##      9   0   0   0   8   3   0  86   1   1   1
```

Conclusion for Logistic Regression Classifier

For Logistic Regression Classifier, training with 8 PCs has a better result than 160 parameters.

4. LASSO Regression Classifier

With 160 parameters

Training a LASSO Regression Classifier

Cross validation is applied to find out the best lambda for prediction and analysis.

```
x = model.matrix(device_category~.,data=train.all)
y = train.all$device_category
model <- cv.glmnet(x, y, family= "multinomial", grouped = TRUE, type.me
asure = "class", alpha = 1, standardize=TRUE)
```

Validation on training dataset

```
predicted <- predict(model, x, type=c("class"), s="lambda.min")
accuracy <- mean(as.character(predicted) == as.character(train.all$devi
ce_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "LASSO")
Type <- c(Type, "Train - 160")
```

Validation accuracy for training dataset

```
cat("The validation accuracy with LASSO Regression Classifier and 160 p
arameters is:", accuracy, "\n")

## The validation accuracy with LASSO Regression Classifier and 160 par
ameters is: 0.855

as.matrix(table(Actual = train.all$device_category, Predicted = predict
ed))

##       Predicted
## Actual   1  10   2   3   4   5   6   7   8   9
##      1  100   0   0   0   0   0   0   0   0   0
##      2    0   3  86   0   0   0  11   0   0   0
##      3    0   0   2  96   0   0   2   0   0   0
##      4    0   0   0   0 100   0   0   0   0   0
##      5    0   1   0   0   0  99   0   0   0   0
##      6    0   7  60   0   0   0  33   0   0   0
##      7    0   0   0   0   0   0   0 100   0   0
##      8    0   0   0   0   0   0   0   0 100   0
##      9    0   0   0   0   0   0   0   0   1  99
##      10   0  42  38   0   0   0  20   0   0   0
```

Prediction on testing dataset

```
predicted <- predict(model, model.matrix(device_category~.,data=test.al
l), type=c("class"), s="lambda.min")
accuracy <- mean(as.character(predicted) == as.character(test.all$devic
e_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "LASSO")
Type <- c(Type, "Test - 160")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with LASSO Regression Classifier and 160 p
arameters is:", accuracy, "\n")

## The prediction accuracy with LASSO Regression Classifier and 160 par
ameters is: 0.1466667
```

```
as.matrix(table(Actual = test.all$device_category, Predicted = predicte
d))
```

```
##       Predicted
## Actual   1  10   2   3   4   6   7   8   9
##      1   1   0   0   0  86   0   0  13   0
##      2   0  21  63   0   0  16   0   0   0
##      3  60   0   0   9  20   0   3   6   2
##      4   0   0   0   0  18   0  34  16  32
##      5  77   0   0   0  23   0   0   0   0
##      6   0  43  30   0   0  27   0   0   0
##      7   0   0   0   0   0   0   0 100   0
##      8  78   0   0   0  11   0   0  11   0
##      9  90   0   0   0   2   0   0   5   3
```

With 8 PCs

Training a LASSO Regression Classifier

```
x = model.matrix(device_category~.,data=train.8pc)
y = train.8pc$device_category
model <- cv.glmnet(x, y, family= "multinomial", grouped = TRUE, type.me
asure = "class", alpha = 1, standardize=TRUE)
```

Validation on training dataset

```
predicted <- predict(model, x, type=c("class"), s="lambda.min")
accuracy <- mean(as.character(predicted) == as.character(train.8pc$devi
ce_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "LASSO")
Type <- c(Type, "Train - 8 PCs")
```

Validation accuracy for training dataset

```
cat("The validation accuracy with LASSO Regression Classifier and 8 PCs
 is:", accuracy,"\n")
```

```
## The validation accuracy with LASSO Regression Classifier and 8 PCs i
s: 0.799
```

```
as.matrix(table(Actual = train.8pc$device_category, Predicted = predict
ed))
```

```
##       Predicted
## Actual  1 10  2  3  4  5  6  7  8  9
##      1 93  0  0  0  2  0  0  0  3  2
##      2  0  0 93  0  0  0  7  0  0  0
##      3  0  0  3 96  0  0  1  0  0  0
##      4  2  0  0  0 94  0  0  3  0  1
##      5  0  1  0  0  0 99  0  0  0  0
```

```
##      6   0  5 76  0  0  0 19  0  0  0
##      7   1  0  0  0  3  0  0 90  5  1
##      8   6  0  0  0  3  0  0  3 85  3
##      9   3  0  0  0  1  0  0  1  2 93
##      10  1 37 48  0  0  0 14  0  0  0
```

Prediction on testing dataset

```
predicted <- predict(model, model.matrix(device_category~.,data=test.8p
c), type=c("class"), s="lambda.min")
accuracy <- mean(as.character(predicted) == as.character(test.8pc$devic
e_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "LASSO")
Type <- c(Type, "Test - 8 PCs")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with LASSO Regression Classifier and 8 PCs
 is:", accuracy,"\n")
```

```
## The prediction accuracy with LASSO Regression Classifier and 8 PCs i
s: 0.22
```

```
as.matrix(table(Actual = test_data$device_category, Predicted = predict
ed))
```

```
##       Predicted
## Actual   1  10   2   4   5   6   7   8   9
##      1   0   0   0  87   0   0   0  13   0
##      2   0  19  52   0   0  26   0   0   3
##      3  34   0  17   2  33   0  12   2   0
##      4   0   0   5  34   0   0  29  32   0
##      5   0   0   0  23  77   0   0   0   0
##      6   0  40  27   0   0  33   0   0   0
##      7   0   0   0 100   0   0   0   0   0
##      8   2   0  10  11  76   0   0   1   0
##      9   0   1   0   8  88   0   1   1   1
```

Conclusion for LASSO Regression Classifier

For LASSO Regression Classifier, training with 8 PCs has a slightly better result than 160 parameters.

5. Ridge Regression Classifier

With 160 parameters

Training a Ridge Regression Classifier

Similar to LASSO regression classifier, cross validation is applied as well for finding the best lambda.

```
x = model.matrix(device_category~.,data=train.all)
y = train.all$device_category
model <- cv.glmnet(x, y, family= "multinomial", grouped = TRUE, type.me
asure = "class", alpha = 0, standardize=TRUE)
```

Validation on training dataset

```
predicted <- predict(model, x, type=c("class"), s="lambda.min")
accuracy <- mean(as.character(predicted) == as.character(train.all$devi
ce_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Ridge")
Type <- c(Type, "Train - 160")
```

Validation accuracy for training dataset

```
cat("The validation accuracy with Ridge Regression Classifier and 160 p
arameters is:", accuracy, "\n")

## The validation accuracy with Ridge Regression Classifier and 160 par
ameters is: 0.812

as.matrix(table(Actual = train.all$device_category, Predicted = predict
ed))

##       Predicted
## Actual  1 10  2  3  4  5  6  7  8  9
##     1  98  0  0  0  2  0  0  0  0  0
##     2   0  0 98  0  0  0  2  0  0  0
##     3   0  0  4 96  0  0  0  0  0  0
##     4   1  0  0  0 99  0  0  0  0  0
##     5   0  1  0  0  0 99  0  0  0  0
##     6   0  5 87  0  0  0  8  0  0  0
##     7   0  0  0  0  0  0  0 95  4  1
##     8   0  0  0  0  0  0  1  2 88  9
##     9   1  0  0  0  2  0  0  1  3 93
##     10  0 38 60  0  0  0  2  0  0  0
```

Prediction on testing dataset

```
predicted <- predict(model, model.matrix(device_category~.,data=test.al
l), type=c("class"), s="lambda.min")
accuracy <- mean(as.character(predicted) == as.character(test.all$devic
e_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Ridge")
Type <- c(Type, "Test - 160")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with Ridge Regression Classifier and 160 p
arameters is:", accuracy, "\n")
```

```
## The prediction accuracy with Ridge Regression Classifier and 160 par
ameters is: 0.2866667
```

```
as.matrix(table(Actual = test.all$device_category, Predicted = predicte
d))
```

```
##       Predicted
## Actual   1  10   2   3   4   5   6   7   8   9
##      1   0   0   0   0  86   1   0   0  13   0
##      2   0  22  78   0   0   0   0   0   0   0
##      3  13   0   0  56  14   6   0   9   2   0
##      4   0   0   0  34  34   0   0   0  32   0
##      5   7   0   0   0  23  70   0   0   0   0
##      6   0  43  49   0   0   0   8   0   0   0
##      7   0   0   0   0 100   0   0   0   0   0
##      8  76   2   0   0   6   0   0   7   9   0
##      9  89   0   0   0   4   0   0   3   1   3
```

With 8 PCs

Training a Ridge Regression Classifier

```
x = model.matrix(device_category~.,data=train.8pc)
y = train.8pc$device_category
model <- cv.glmnet(x, y, family= "multinomial", grouped = TRUE, type.me
asure = "class", alpha = 0, standardize=TRUE)
```

Validation on training dataset

```
predicted <- predict(model, x, type=c("class"), s="lambda.min")
accuracy <- mean(as.character(predicted) == as.character(train.8pc$devi
ce_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Ridge")
Type <- c(Type, "Train - 8 PCs")
```

Validation accuracy for training dataset

```
cat("The validation accuracy with Ridge Regression Classifier and 8 PCs
 is:", accuracy,"\n")
```

```
## The validation accuracy with Ridge Regression Classifier and 8 PCs i
s: 0.672
```

```
as.matrix(table(Actual = train.8pc$device_category, Predicted = predict
ed))
```

```
##        Predicted
## Actual  1 10  2  3  4  5  6  7  8  9
##      1  79  0  0  0  6  0  0  0  0 15
##      2   0  0 98  0  0  0  2  0  0  0
##      3   0  0  5 95  0  0  0  0  0  0
##      4  30  0  0  0 18  0  0 51  0  1
##      5   3  0  0  0  0 97  0  0  0  0
##      6   0 11 87  0  0  0  2  0  0  0
##      7   0  0  0  0  0  0  0 94  4  2
##      8   6  0  0  0  1  0  0 11 78  4
##      9   5  0  0  0  1  0  0  7  8 79
##      10  7 32 60  0  0  1  0  0  0  0
```

Prediction on testing dataset

```
predicted <- predict(model, model.matrix(device_category~.,data=test.8p
c), type=c("class"), s="lambda.min")
accuracy <- mean(as.character(predicted) == as.character(test.8pc$devic
e_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "Ridge")
Type <- c(Type, "Test - 8 PCs")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with Ridge Regression Classifier and 8 PCs
 is:", accuracy,"\n")

## The prediction accuracy with Ridge Regression Classifier and 8 PCs i
s: 0.3811111

as.matrix(table(Actual = test_data$device_category, Predicted = predict
ed))

##        Predicted
## Actual  1 10  2  3  4  5  6  7  8  9
##      1  0  0  0  0  0 84  0  0 13  3
##      2  0 22 78  0  0  0  0  0  0  0
##      3  1  0  0 58  1 26  0 11  2  1
##      4  2  0  0 34  0 28  0  0 32  4
##      5  0  0  0  0  0 94  0  6  0  0
##      6  0 48 49  0  0  0  3  0  0  0
##      7  0  0  0  0  0  1  0 99  0  0
##      8  1  0  0  0  2  7  0 80  8  2
##      9  0  0  0  0  2  7  0 88  0  3
```

Conclusion for Ridge Regression Classifier

For Ridge Regression Classifier, training with 8 PCs has a slightly better result than 160 parameters.

6. XGBoost Classifier

With 160 parameters

Training a XGBoost Classifier

```
x = model.matrix(device_category~.,data=train.all)
cat = train.all$device_category
y = as.integer(cat)-1
xgb.train <- xgb.DMatrix(data = x, label = y)
xgb.test <- xgb.DMatrix(data = model.matrix(device_category~.,data=test
.all), label = as.integer(test.all$device_category)-1)

num_class <- length(unique(y))
params = list(
  booster="gbtree",
  eta=0.001,
  max_depth=5,
  gamma=3,
  subsample=0.75,
  colsample_bytree=1,
  objective="multi:softprob",
  eval_metric="mlogloss",
  num_class=num_class
)


model<- xgb.train(
  params=params,
  data=xgb.train,
  nrounds=10000,
  nthreads=1,
  early_stopping_rounds=10,
  watchlist=list(val1=xgb.train,val2=xgb.test),
  verbose=0
)
```

Validation on training dataset

```
predicted <- predict(model, x, reshape=T)
predicted <- as.data.frame(predicted)
colnames(predicted) <- unique(cat)
predicted$prediction <- apply(predicted,1,function(x) colnames(predicte
d)[which.max(x)])
predicted$label <- unique(cat)[y+1]
accuracy <- mean(predicted$prediction == predicted$label)

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "XGBoost")
Type <- c(Type, "Train - 160")
```

Validation accuracy for training dataset

```
cat("The validation accuracy with XGBoost Classifier and 160 parameters
 is:", accuracy, "\n")
```

```
## The validation accuracy with XGBoost Classifier and 160 parameters i
s: 0.903
```

```
as.matrix(table(Actual = predicted$label, Predicted = predicted$predict
ion))
```

```
##       Predicted
## Actual   1  10   2   3   4   5   6   7   8   9
##     1   96   0   1   3   0   0   0   0   0   0
##     2    1   0  99   0   0   0   0   0   0   0
##     3    0   0   3  97   0   0   0   0   0   0
##     4    0   0   0   1  99   0   0   0   0   0
##     5    0   0   0   0   1  95   2   0   2   0
##     6    0   0   0   0   0   0  73   0  19   8
##     7    0   0   0   0   0   0   0 100   0   0
##     8    0   0   0   0   0   0  10   1  81   8
##     9    0   0   0   0   0   0  11   0  23  66
##     10   0  97   0   0   0   2   1   0   0   0
```

Prediction on testing dataset

```
predicted <- predict(model, model.matrix(device_category~.,data=test.al
l), reshape=T)
predicted <- as.data.frame(predicted)
colnames(predicted) <- unique(cat)
predicted$prediction <- apply(predicted,1,function(x) colnames(predicte
d)[which.max(x)])
predicted$label <- unique(cat)[as.integer(test.all$device_category)]
accuracy <- mean(predicted$prediction == predicted$label)

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "XGBoost")
Type <- c(Type, "Test - 160")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with XGBoost Classifier and 160 parameters
 is:", accuracy, "\n")
```

```
## The prediction accuracy with XGBoost Classifier and 160 parameters i
s: 0.64
```

```
as.matrix(table(Actual = predicted$label, Predicted = predicted$predict
ion))
```

```
##       Predicted
## Actual   1  10   2   3   4   5   6   7   8   9
```

```
##      1   90   0   0  10   0   0   0   0   0   0
##      2    4   0  54  42   0   0   0   0   0   0
##      3    0   0  21  71   2   2   0   4   0   0
##      4    4   0  10   0  85   0   0   1   0   0
##      5    2   3  21   7   0  67   0   0   0   0
##      6    0   0   0   0   0   0   0   0   0   0
##      7   39   0  26   0   0   1   0  34   0   0
##      8    0   0   0   0   0   0  33   0  49  18
##      9    0   0   0   0   0   0  53   0  21  26
##      10   0 100   0   0   0   0   0   0   0   0
```

With 8 PCs

Training a XGBoost Classifier

```
x = model.matrix(device_category~.,data=train.8pc)
cat = train.8pc$device_category
y = as.integer(cat)-1
xgb.train <- xgb.DMatrix(data = x, label = y)
xgb.test <- xgb.DMatrix(data = model.matrix(device_category~.,data=test
.8pc), label = as.integer(test.8pc$device_category)-1)

num_class <- length(unique(y))
params = list(
  booster="gbtree",
  eta=0.001,
  max_depth=5,
  gamma=3,
  subsample=0.75,
  colsample_bytree=1,
  objective="multi:softprob",
  eval_metric="mlogloss",
  num_class=num_class
)


model<- xgb.train(
  params=params,
  data=xgb.train,
  nrounds=10000,
  nthreads=1,
  early_stopping_rounds=10,
  watchlist=list(val1=xgb.train,val2=xgb.test),
  verbose=0
)
```

Validation on training dataset

```
predicted <- predict(model, x, reshape=T)
predicted <- as.data.frame(predicted)
```

```
colnames(predicted) <- unique(cat)
predicted$prediction <- apply(predicted,1,function(x) colnames(predicte
d)[which.max(x)])
predicted$label <- unique(cat)[y+1]
accuracy <- mean(predicted$prediction == predicted$label)

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "XGBoost")
Type <- c(Type, "Train - 8 PCs")
```

Validation accuracy for training dataset

```
cat("The validation accuracy with XGBoost Classifier and 8 PCs is:", ac
curacy, "\n")

## The validation accuracy with XGBoost Classifier and 8 PCs is: 0.872

as.matrix(table(Actual = predicted$label, Predicted = predicted$predict
ion))

##       Predicted
## Actual  1 10  2  3  4  5  6  7  8  9
##     1  93  0  1  3  2  0  0  1  0  0
##     2   0  0 97  3  0  0  0  0  0  0
##     3   2  1  3 92  1  1  0  0  0  0
##     4   0  0  5  0 95  0  0  0  0  0
##     5   0  1  0  0  0 95  2  0  1  1
##     6   0  0  0  0  0  0 70  0 20 10
##     7   1  0  1  0  0  0  0 98  0  0
##     8   0  0  0  0  0  0 10  0 76 14
##     9   0  0  0  0  0  0 11  0 32 57
##     10  0 99  0  0  0  0  1  0  0  0
```

Prediction on testing dataset

```
predicted <- predict(model, model.matrix(device_category~.,data=test.8p
c), reshape=T)
predicted <- as.data.frame(predicted)
colnames(predicted) <- unique(cat)
predicted$prediction <- apply(predicted,1,function(x) colnames(predicte
d)[which.max(x)])
predicted$label <- unique(cat)[as.integer(test.8pc$device_category)]
accuracy <- mean(predicted$prediction == predicted$label)

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "XGBoost")
Type <- c(Type, "Test - 8 PCs")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with XGBoost Classifier and 8 PCs is:", ac
curacy, "\n")
```

```
## The prediction accuracy with XGBoost Classifier and 8 PCs is: 0.2355
556
```

```
as.matrix(table(Actual = predicted$label, Predicted = predicted$predict
ion))
```

```
##       Predicted
## Actual   1   2   3   4   5   6   7   9
##     1    0   0   0 100   0   0   0   0
##     2    0   3   3  92   0   0   2   0
##     3    0   1   8  89   0   0   2   0
##     4    0  13   0  87   0   0   0   0
##     5    1   3   0  35  58   0   3   0
##     6    0   0   0   0   0   0   0   0
##     7    0  32   0  34  34   0   0   0
##     8    0   0   0   0   0  28   0  72
##     9    0   0   0   0   0  44   0  56
##     10   0   0   0 100   0   0   0   0
```

Conclusion for XGBoost Classifier

For XGBoost Classifier, training with 160 parameters is much better than 8 PCs.

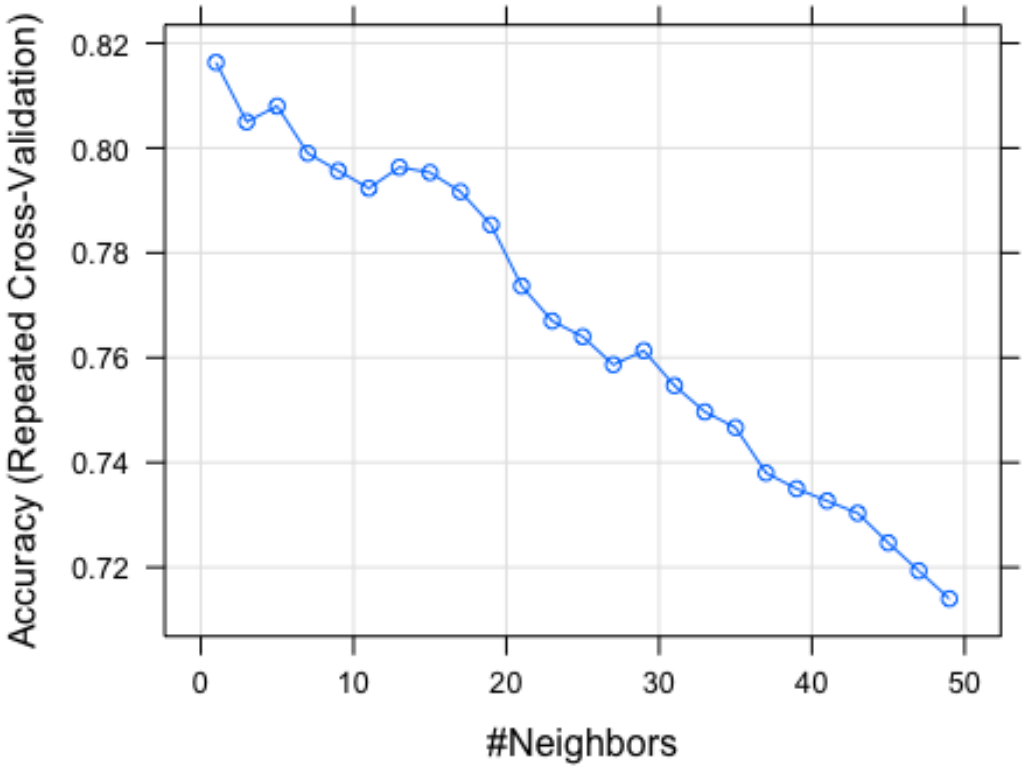
7. K-nearest Neighbors Classifier

With 160 parameters

Training a K-nearest Neighbors Classifier

Some articles suggest that 'k' can be set to sqrt(n), but there is just a begining trial and the best way to determine 'k' is through validation. As a result, cross validation is applied with every odd number between 1 to 50. From the output result, it is found that the best k to be used is 1.

```
grid=expand.grid(k=seq(1,50,by=2))
ctrl <- trainControl(method='repeatedcv', repeats = 3,search = 'grid')
model <- train(device_category~.,data=train.all, method = 'knn', trCont
rol = ctrl, tuneGrid = grid, preProcess =c('center','scale'),tuneLength
 = 20)
plot(model)
```



Validation on training dataset

```
predicted  <- knn(train.all,train.all,cl=train.all$device_category,k=1)
accuracy <- mean(as.character(predicted) == as.character(train.all$devi
ce_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "KNN")
Type <- c(Type, "Train - 160")
```

Validation accuracy for training dataset

```
cat("The validation accuracy with KNN Classifier and 160 parameters is:
", accuracy, "\n")

## The validation accuracy with KNN Classifier and 160 parameters is: 1

as.matrix(table(Actual = train.all$device_category, Predicted = predict
ed))

##       Predicted
## Actual   1   2   3   4   5   6   7   8   9  10
##      1  100   0   0   0   0   0   0   0   0   0
##      2    0 100   0   0   0   0   0   0   0   0
```

```
##      3    0   0 100   0   0   0   0   0   0   0
##      4    0   0   0 100   0   0   0   0   0   0
##      5    0   0   0   0 100   0   0   0   0   0
##      6    0   0   0   0   0 100   0   0   0   0
##      7    0   0   0   0   0   0 100   0   0   0
##      8    0   0   0   0   0   0   0 100   0   0
##      9    0   0   0   0   0   0   0   0 100   0
##      10   0   0   0   0   0   0   0   0   0 100
```

Prediction on testing dataset

```
predicted  <- knn(train.all,test.all,cl=train.all$device_category,k=1)
accuracy <- mean(as.character(predicted) == as.character(test.all$devic
e_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "KNN")
Type <- c(Type, "Test - 160")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with KNN Classifier and 160 parameters is:
", accuracy, "\n")

## The prediction accuracy with KNN Classifier and 160 parameters is: 0
.7333333

as.matrix(table(Actual = test.all$device_category, Predicted = predicte
d))

##        Predicted
## Actual   1   2   3   4   5   6   7   8   9  10
##      1 100   0   0   0   0   0   0   0   0   0
##      2   0  83   0   0   0   2   0   0   0  15
##      3  12   0  63   6   1   0   4   6   8   0
##      4   0   0  34  34   0   0   0  32   0   0
##      5   0   0   0   0 100   0   0   0   0   0
##      6   0   0   0   0   0  72   0   0   0  28
##      7   0   0   0   0   0   0  99   1   0   0
##      8   0   0   2   0   0   0   1  47  12  38
##      9   0   0   0   0   0   0   2  36  62   0
```

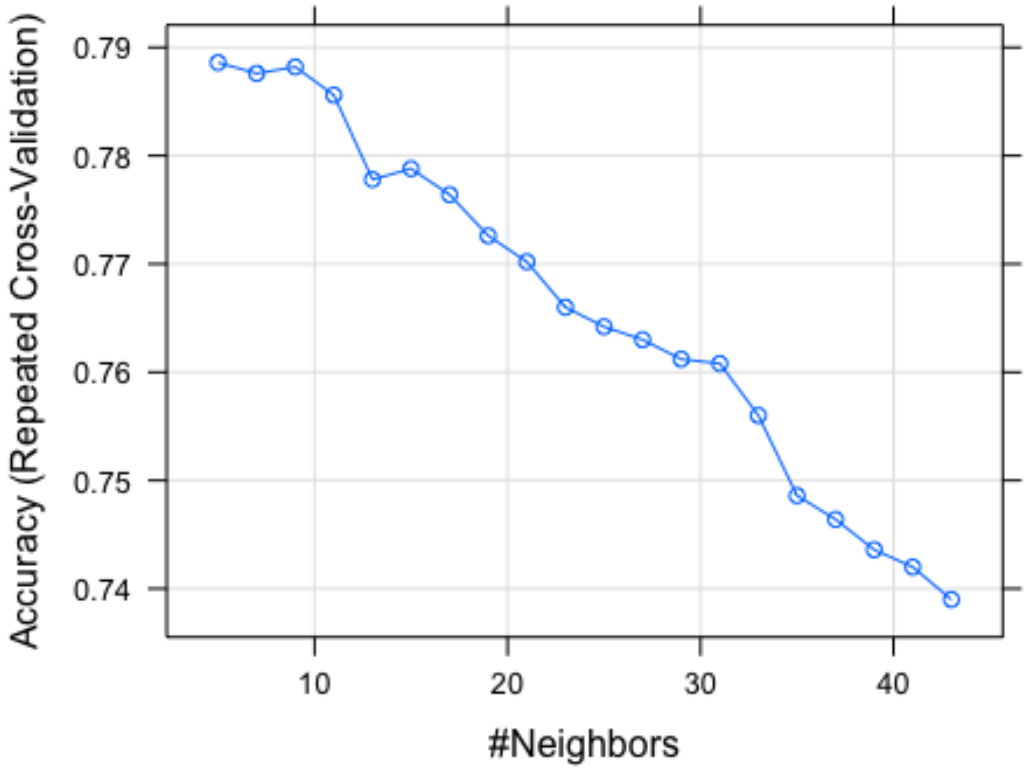
With 8 PCs

Training a K-nearest Neighbors Classifier

Similarly, to determine number of 'k' to be used, cross validation is applied. From the output result, it is found that the best k to be used is 7.

```
ctrl <- trainControl(method='repeatedcv',repeats = 5,search = 'grid')
model <- train(device_category~.,data=train.8pc, method = 'knn', trCont
```

```
rol = ctrl, preProcess =c('center','scale'),tuneLength = 20)
plot(model)
```



Validation on training dataset

```
predicted  <- knn(train.8pc,train.8pc,cl=train.8pc$device_category,k=7)
accuracy <- mean(predicted == train.8pc$device_category)

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "KNN")
Type <- c(Type, "Train - 8 PCs")
```

Validation accuracy for training dataset

```
cat("The validation accuracy with KNN Classifier and 8 PCs is:", accura
cy, "\n")

## The validation accuracy with KNN Classifier and 8 PCs is: 0.979

as.matrix(table(Actual = train.8pc$device_category, Predicted = predict
ed))

##       Predicted
## Actual   1   2   3   4   5   6   7   8   9  10
```

```
##      1  100   0   0   0   0   0   0   0   0   0
##      2    0 100   0   0   0   0   0   0   0   0
##      3    0   2  98   0   0   0   0   0   0   0
##      4    1   0   0  99   0   0   0   0   0   0
##      5    0   0   1   0  97   0   0   2   0   0
##      6    0   0   0   0   0 100   0   0   0   0
##      7    0   0   1   0   0   0  95   2   2   0
##      8    0   0   1   0   0   0   0  94   5   0
##      9    0   0   0   0   0   0   0   4  96   0
##      10   0   0   0   0   0   0   0   0   0 100
```

Prediction on testing dataset

```
predicted  <- knn(train.8pc,test.8pc,cl=train.8pc$device_category,k=7)
accuracy <- mean(as.character(predicted)== as.character(test.8pc$device
_category))

Accuracy <- c(Accuracy, accuracy)
Model <- c(Model, "KNN")
Type <- c(Type, "Test - 8 PCs")
```

Prediction accuracy for testing dataset

```
cat("The prediction accuracy with KNN Classifier and 8 PCs is:", accura
cy, "\n")

## The prediction accuracy with KNN Classifier and 8 PCs is: 0.4444444

as.matrix(table(Actual = test.8pc$device_category, Predicted = predicte
d))

##       Predicted
## Actual   1   2   3   4   5   6   7   8   9  10
##      1  95   0   0   0   0   0   0   4   1   0
##      2   0  78   0   0   0   7   0   0   0  15
##      3   3   0  59   1  31   0   0   4   2   0
##      4   2   0  34   0   0   0   0   0  64   0
##      5   0   0   0   0  77   0   0   0  23   0
##      6   0   0   0   0   0  67   0   0   0  33
##      7   0   0   0   0   0   0   0   0 100   0
##      8   0   0   0   0  76   0   0  11  13   0
##      9   0   0   0   0  87   0   0   0  13   0
```

Conclusion for KNN Classifier

For KNN Classifier, training with 160 parameters is better than 8 PCs.

Summary

From the modeling result, KNN has the highest validation and prediction accuracy for both input of 160 parameters and 8PCs. Both inputs have a validation accuracy more than 95% and the prediction accuracy can reach 73% when using 160 parameters as input. As for 8PCs, it has around 44% accuracy.

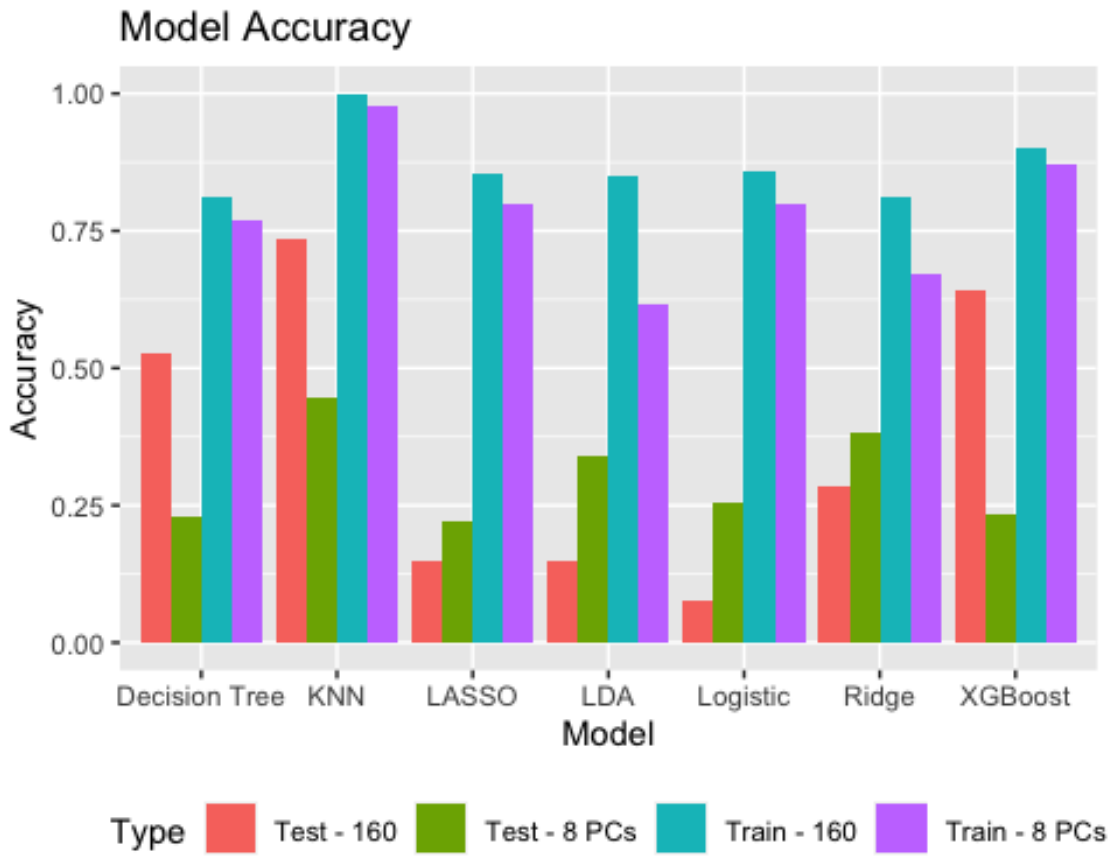
XGBoost comes the second. Both inputs have the validation accuracy more than 85% and the prediction accuracy can reach 64% with 160 parameters. However, when using 8PCs as input, the prediction accuracy drops to 24%. Although the gap between validation accuracy for both inputs is small, the prediction accuracy can have a big difference!

The third highest prediction accuracy goes to Decision Tree. Both inputs have validation accuracy around 77% - 80%. But the prediction accuracy is around 53% with 160 parameters. Similar to XGBoost, the prediction accuracy for 8PCs is only 23% which is less than half of the prediction accuracy with 160 parameters.

Prediction with Multi-class LDA, LASSO, Ridge and Logistic regression are not desired. The prediction accuracy for both 160 parameters and 8 PCs are less than 30%. The worst goes to Logistics regression with 160 parameters, which is 7% only! Both Multi-class LDA and LASSO have the prediction accuracy around 15% and Ridge has around 30% with 160 parameters. It is interesting that for these four models, the prediction results with 8PCs are better than 160 parameters, although they are still not desired. They obtain at 34%, 22%, 38% and 26% for Multi-class LDA, LASSO, Ridge and Logistic regression respectively. Recall from the correlation heatmap, there are some parameters which are highly correlated. The poor predication with 160 parameters maybe a result of collinearity.

From here, we can conclude that for this dataset, nonparametric models seem to have a better result than parametric models. In addition, using all 160 parameters obtain a better prediction accuracy than 8 PCs for nonparametric models, and versa vice for parametric models.

```
summary_df <- data.frame(Accuracy, Model, Type)
ggplot(summary_df, aes(Model, Accuracy, fill = Type)) +
  geom_bar(stat="identity", position = "dodge") +
  labs(title="Model Accuracy") + theme(legend.position="bottom")
```



Moreover, PCA is useful to speed up the computation by reducing the dimensionality of the dataset. However, the result of this project shows that losing just 10% variances may result in a 50% worse of prediction accuracy when comparing to all parameters. It really varies among different models and different datasets.

Evaluation

Regardless on the classifiers being used, the overall prediction performance is unsatisfactory as the prediction accuracy for all classifiers are not high. After evaluation, it is found that the number of dummy columns (i.e. columns with single value) in testing dataset is far less than the training dataset. The removal of dummy columns in the data preparation part has removed more columns than expected when predicting with testing dataset. It seems that the training dataset may not be representative enough.

A solution to solve this issue is to combine the training and testing dataset, then randomly select training and testing samples from the full combined dataset. However, we believe it is not suitable to make such move as the training and testing

dataset have already been defined by the dataset provider, even though we discovered such problem in the training dataset during model training.

The randomness of training and testing dataset should be emphasized in this exercise and when doing prediction analysis.

```
dummy_columns_train <- vapply(train_data_in, function(x) length(unique(
x)) <= 1, logical(1L))
cat("The number of dummy columns in training dataset is ", length(train
_data_in[dummy_columns_train]))

## The number of dummy columns in training dataset is  137

dummy_columns_test <- vapply(test_data_in, function(x) length(unique(x)
) <= 1, logical(1L))
cat("\nThe number of dummy columns in testing dataset is ", length(test
_data_in[dummy_columns_test]))

##
## The number of dummy columns in testing dataset is  47
```

Reference

submartingale. (2017, 6 5). *Too Many Weights in Multinomial logistic regression and the code is running for hours* . Retrieved from stackoverflow: https://stackoverflow.com/questions/36303404/too-many-weights-in-multinomial-logistic-regression-and-the-code-is-running-for