> rm(list = ls())

> setwd("/Users/haonguyen/Documents/GitHub/BigDataAnalyticFinalProject")

>

> # install.packages("class")

> library(class)

> # install.packages("e1071")

> library(e1071)

> library(lattice)

> library(ggplot2)

> library(caret)

> # install.packages("DMwR")

> library(DMwR) # For KNN

> # install.packages("ggvis")

> library(ggvis)

> library(dplyr)

> library(randomForest)

> # install.packages("devtools")

> library(devtools)

> #devtools::install\_github('araastat/reprtree')

> library(reprtree)

> # install.packages("mlbench")

> library(mlbench)

> # install.packages("caret")

> library(caret)

> # install.packages("caretEnsemble")

> library(caretEnsemble)

> library(tidyverse)

>

>

>

> ##---------------------------------IMPORT DATA---------------------------------

> #Original dataset has splitted by 70% training and 30% testing

> X\_train<-read.table("./UCI HAR Dataset/train/X\_train.txt")

> y\_train<-read.table("./UCI HAR Dataset/train/y\_train.txt")

> X\_test<-read.table("./UCI HAR Dataset/test/X\_test.txt")

> y\_test<-read.table("./UCI HAR Dataset/test/y\_test.txt")

>

> #To understand what is each column associated with what.

> features<-read.table("./UCI HAR Dataset/features.txt")

> #----------------------------------Data Visualization----------------------------

>

> ## PIE CHART TO SHOW DISTRIBUTION OF ALL SIX ACTIVITIES IN TRAINING RESULTS

>

> df\_activity <- y\_train

>

> df\_activity <- df\_activity %>%

+ mutate(Activity = case\_when(V1 == 1 ~ 'WALKING',

+ V1 == 2 ~ 'WALKING\_UPSTAIRS',

+ V1 == 3 ~ 'WALKING\_DOWNSTAIRS',

+ V1 == 4 ~ 'SITTING',

+ V1 == 5 ~ 'STANDING',

+ V1 == 6 ~ 'LAYING'))

>

>

> theme\_set(theme\_classic())

>

> pie <- ggplot(df\_activity, aes(x = "", fill = factor(Activity))) +

+ geom\_bar(width = 1) +

+ theme(axis.line = element\_blank(),

+ plot.title = element\_text(hjust=0.5)) +

+ labs(fill="Activity",

+ x=NULL,

+ y=NULL,

+ title="Pie Chart of Activity - TRAINING",

+ caption="Source: y\_train")

>

> pie + coord\_polar(theta = "y", start=0)

>

> #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

> # To check tBodyAcc-max()-XYZ values are between -1 and +1

> #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

>

> df\_filtered\_features <- features %>%

+ mutate(Col\_Names = paste("V",features$V1, sep = "")) %>%

+ filter(str\_detect(V2, pattern = "max()")) %>%

+ select(Col\_Names, V2)

>

> X\_train\_max <- X\_train %>%

+ select(c(df\_filtered\_features$Col\_Names))

>

> ggplot(X\_train\_max, aes(x=(1:7352))) +

+ geom\_point(mapping = aes(y = X\_train\_max$V10), color = 'blue') +

+ geom\_point(mapping = aes(y = X\_train\_max$V11), color = 'black') +

+ geom\_point(mapping = aes(y = X\_train\_max$V12), color = 'red') +

+ labs(x = "Observations",y="Values" , title="tBodyAcc-max()-XYZ value plot")

>

> #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

> # To check all angle() values are between -1 and +1

> #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

> df\_filtered\_features\_2 <- features %>%

+ mutate(Col\_Names = paste("V",features$V1, sep = "")) %>%

+ filter(str\_detect(V2, pattern = "angle()")) %>%

+ select(Col\_Names, V2)

>

> X\_train\_angle <- X\_train %>%

+ select(c(df\_filtered\_features\_2$Col\_Names))

>

> ggplot(X\_train\_angle, aes(x=(1:7352))) +

+ geom\_point(mapping = aes(y = X\_train\_angle$V555), color = 'blue') +

+ geom\_point(mapping = aes(y = X\_train\_angle$V556), color = 'black') +

+ geom\_point(mapping = aes(y = X\_train\_angle$V557), color = 'red') +

+ geom\_point(mapping = aes(y = X\_train\_angle$V558), color = 'green') +

+ geom\_point(mapping = aes(y = X\_train\_angle$V559), color = 'magenta') +

+ geom\_point(mapping = aes(y = X\_train\_angle$V560), color = 'yellow') +

+ labs(x = "Observations", y="Values" , title="angle() values plot")

> ##------------------------------DATA PREPROCESSING-----------------------------

>

> #Change column name in label of training and testing

> y.train <- as\_tibble(y\_train)

Warning message:

In dontCheck(fnname) : reached elapsed time limit

> y.train = y.train %>%

+ rename(

+ y = V1

+ )

> dim(y\_train)

[1] 7352 1

>

> y.test <- as\_tibble(y\_test)

> y.test = y.test %>%

+ rename(

+ y = V1

+ )

>

> dim(y.test)

[1] 2947 1

>

> #Combine

> train.df = data.frame(X\_train, y.train)

> valid.df = data.frame(X\_test, y.test)

>

>

> ## Check null value

> sum(is.na(train.df))

[1] 0

> sum(is.na(valid.df))

[1] 0

>

>

> #Check redundant data

> res\_train <- cor(X\_train)

> res\_test <- cor(X\_test)

> #----------------------------------Without Ensembles------------------------------------

> ##---------------------------------KNN---------------------------------

> # \*\*\*\*\*Need to try rectangular, triangular, etc.

> #------1st train---------

> ## A 5-nearest neighbors model with no normalization

> knn5\_pred <- knn(train = train.df, test = valid.df, cl = train.df$y, k=5)

> #NROW(knn5\_pred) # compare size with y\_test

> ##Evaluation KNN\_5

> table(knn5\_pred , valid.df$y)

knn5\_pred 1 2 3 4 5 6

1 489 22 3 0 0 0

2 1 444 35 2 0 0

3 6 5 382 0 0 0

4 0 0 0 455 6 0

5 0 0 0 34 526 1

6 0 0 0 0 0 536

> confusionMatrix(table(knn5\_pred , valid.df$y))

Confusion Matrix and Statistics

knn5\_pred 1 2 3 4 5 6

1 489 22 3 0 0 0

2 1 444 35 2 0 0

3 6 5 382 0 0 0

4 0 0 0 455 6 0

5 0 0 0 34 526 1

6 0 0 0 0 0 536

Overall Statistics

Accuracy : 0.961

95% CI : (0.9533, 0.9677)

No Information Rate : 0.1822

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9531

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6

Sensitivity 0.9859 0.9427 0.9095 0.9267 0.9887 0.9981

Specificity 0.9898 0.9847 0.9956 0.9976 0.9855 1.0000

Pos Pred Value 0.9514 0.9212 0.9720 0.9870 0.9376 1.0000

Neg Pred Value 0.9971 0.9890 0.9851 0.9855 0.9975 0.9996

Prevalence 0.1683 0.1598 0.1425 0.1666 0.1805 0.1822

Detection Rate 0.1659 0.1507 0.1296 0.1544 0.1785 0.1819

Detection Prevalence 0.1744 0.1636 0.1334 0.1564 0.1904 0.1819

Balanced Accuracy 0.9878 0.9637 0.9526 0.9621 0.9871 0.9991

>

> #------2nd train---------

> ## A 10-nearest neighbors model with no normalization

> knn10\_pred <- knn(train = train.df, test = valid.df, cl = train.df$y, k=10)

> ##Evaluation KNN\_10å

> table(knn10\_pred , valid.df$y)

knn10\_pred 1 2 3 4 5 6

1 489 18 1 0 0 0

2 1 451 42 2 0 0

3 6 2 377 0 0 0

4 0 0 0 456 6 0

5 0 0 0 33 526 1

6 0 0 0 0 0 536

> confusionMatrix(table(knn10\_pred , valid.df$y))

Confusion Matrix and Statistics

knn10\_pred 1 2 3 4 5 6

1 489 18 1 0 0 0

2 1 451 42 2 0 0

3 6 2 377 0 0 0

4 0 0 0 456 6 0

5 0 0 0 33 526 1

6 0 0 0 0 0 536

Overall Statistics

Accuracy : 0.962

95% CI : (0.9544, 0.9686)

No Information Rate : 0.1822

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9543

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6

Sensitivity 0.9859 0.9575 0.8976 0.9287 0.9887 0.9981

Specificity 0.9922 0.9818 0.9968 0.9976 0.9859 1.0000

Pos Pred Value 0.9626 0.9093 0.9792 0.9870 0.9393 1.0000

Neg Pred Value 0.9971 0.9918 0.9832 0.9859 0.9975 0.9996

Prevalence 0.1683 0.1598 0.1425 0.1666 0.1805 0.1822

Detection Rate 0.1659 0.1530 0.1279 0.1547 0.1785 0.1819

Detection Prevalence 0.1724 0.1683 0.1306 0.1568 0.1900 0.1819

Balanced Accuracy 0.9891 0.9697 0.9472 0.9631 0.9873 0.9991

>

> #------3rd train---------

> ## A 25-nearest neighbors model with no normalization

> knn25\_pred <- knn(train = train.df, test = valid.df, cl = train.df$y, k=25)

> ##Evaluation KNN\_25

> table(knn25\_pred , valid.df$y)

knn25\_pred 1 2 3 4 5 6

1 495 18 2 0 0 0

2 0 452 43 3 0 0

3 1 1 375 0 0 0

4 0 0 0 457 2 0

5 0 0 0 31 530 1

6 0 0 0 0 0 536

> confusionMatrix(table(knn25\_pred , valid.df$y))

Confusion Matrix and Statistics

knn25\_pred 1 2 3 4 5 6

1 495 18 2 0 0 0

2 0 452 43 3 0 0

3 1 1 375 0 0 0

4 0 0 0 457 2 0

5 0 0 0 31 530 1

6 0 0 0 0 0 536

Overall Statistics

Accuracy : 0.9654

95% CI : (0.9581, 0.9717)

No Information Rate : 0.1822

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9584

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6

Sensitivity 0.9980 0.9597 0.8929 0.9308 0.9962 0.9981

Specificity 0.9918 0.9814 0.9992 0.9992 0.9867 1.0000

Pos Pred Value 0.9612 0.9076 0.9947 0.9956 0.9431 1.0000

Neg Pred Value 0.9996 0.9922 0.9825 0.9863 0.9992 0.9996

Prevalence 0.1683 0.1598 0.1425 0.1666 0.1805 0.1822

Detection Rate 0.1680 0.1534 0.1272 0.1551 0.1798 0.1819

Detection Prevalence 0.1748 0.1690 0.1279 0.1558 0.1907 0.1819

Balanced Accuracy 0.9949 0.9705 0.9460 0.9650 0.9915 0.9991

> #------------------------

> # \*\*\* Not such different in change in K-neighbor 5/10/25

> # \*\*\* Class 3 and 4 did not have a google accuracy. Best classes are 1 and 6.

>

> #Accuracy ~ 90%

> fit <-svm(y ~ ., data=train.df)

> summary(fit)

Call:

svm(formula = y ~ ., data = train.df)

Parameters:

SVM-Type: C-classification

SVM-Kernel: radial

cost: 1

Number of Support Vectors: 2330

( 518 571 234 300 358 349 )

Number of Classes: 6

Levels:

1 2 3 4 5 6

> summary(fit)

Call:

svm(formula = y ~ ., data = train.df)

Parameters:

SVM-Type: C-classification

SVM-Kernel: radial

cost: 1

Number of Support Vectors: 2330

( 518 571 234 300 358 349 )

Number of Classes: 6

Levels:

1 2 3 4 5 6

> ## predict class: Training

> svm\_pred\_train <- predict(fit, newdata = train.df[,-562])

> confusionMatrix(svm\_pred\_train, train.df$y)

Confusion Matrix and Statistics

Reference

Prediction 1 2 3 4 5 6

1 1226 0 0 0 0 0

2 0 1073 0 0 0 0

3 0 0 986 0 0 0

4 0 0 0 1244 56 0

5 0 0 0 42 1318 0

6 0 0 0 0 0 1407

Overall Statistics

Accuracy : 0.9867

95% CI : (0.9838, 0.9892)

No Information Rate : 0.1914

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.984

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6

Sensitivity 1.0000 1.0000 1.0000 0.9673 0.9592 1.0000

Specificity 1.0000 1.0000 1.0000 0.9908 0.9930 1.0000

Pos Pred Value 1.0000 1.0000 1.0000 0.9569 0.9691 1.0000

Neg Pred Value 1.0000 1.0000 1.0000 0.9931 0.9907 1.0000

Prevalence 0.1668 0.1459 0.1341 0.1749 0.1869 0.1914

Detection Rate 0.1668 0.1459 0.1341 0.1692 0.1793 0.1914

Detection Prevalence 0.1668 0.1459 0.1341 0.1768 0.1850 0.1914

Balanced Accuracy 1.0000 1.0000 1.0000 0.9791 0.9761 1.0000

> ## predict class: Validation

> svm\_pred\_valid <- predict(fit, newdata = valid.df[,-562])

> confusionMatrix(svm\_pred\_valid, valid.df$y)

Confusion Matrix and Statistics

Reference

Prediction 1 2 3 4 5 6

1 482 14 6 0 0 0

2 6 456 28 1 0 0

3 8 1 386 0 0 0

4 0 0 0 441 29 0

5 0 0 0 47 503 0

6 0 0 0 2 0 537

Overall Statistics

Accuracy : 0.9518

95% CI : (0.9435, 0.9593)

No Information Rate : 0.1822

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9421

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6

Sensitivity 0.9718 0.9682 0.9190 0.8982 0.9455 1.0000

Specificity 0.9918 0.9859 0.9964 0.9882 0.9805 0.9992

Pos Pred Value 0.9602 0.9287 0.9772 0.9383 0.9145 0.9963

Neg Pred Value 0.9943 0.9939 0.9867 0.9798 0.9879 1.0000

Prevalence 0.1683 0.1598 0.1425 0.1666 0.1805 0.1822

Detection Rate 0.1636 0.1547 0.1310 0.1496 0.1707 0.1822

Detection Prevalence 0.1703 0.1666 0.1340 0.1595 0.1866 0.1829

Balanced Accuracy 0.9818 0.9770 0.9577 0.9432 0.9630 0.9996

> ## Train a best linear-kernel support vector classifier

> linear.tune <- tune.svm(y ~ ., data=train.df,

+ kernel = "linear",

+ cost = c(0.001, 0.01, 0.1, 1, 5, 10)) #Cost is the price of the misclassification ( from 0.1 (10^(-1)) to 100 (10^2) in multiples of 10.)

> summary(poly.tune)

Error in summary(poly.tune) : object 'poly.tune' not found

> summary(linear.tune)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost

0.1

- best performance: 0.01400898

- Detailed performance results:

cost error dispersion

1 1e-03 0.02896942 0.007071655

2 1e-02 0.02040280 0.005913592

3 1e-01 0.01400898 0.004347999

4 1e+00 0.01591319 0.004104342

5 5e+00 0.01822575 0.005095708

6 1e+01 0.01836217 0.003965099

> best.linear <- linear.tune$best.model

> linear.pred.train <- predict(best.linear, newdata = train.df)

> confusionMatrix(poly.pred.train, train.df$y)

Error in confusionMatrix(poly.pred.train, train.df$y) :

object 'poly.pred.train' not found

> confusionMatrix(linear.pred.train, train.df$y)

Confusion Matrix and Statistics

Reference

Prediction 1 2 3 4 5 6

1 1226 0 0 0 0 0

2 0 1073 0 0 0 0

3 0 0 986 0 0 0

4 0 0 0 1259 27 0

5 0 0 0 27 1347 0

6 0 0 0 0 0 1407

Overall Statistics

Accuracy : 0.9927

95% CI : (0.9904, 0.9945)

No Information Rate : 0.1914

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9912

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6

Sensitivity 1.0000 1.0000 1.0000 0.9790 0.9803 1.0000

Specificity 1.0000 1.0000 1.0000 0.9955 0.9955 1.0000

Pos Pred Value 1.0000 1.0000 1.0000 0.9790 0.9803 1.0000

Neg Pred Value 1.0000 1.0000 1.0000 0.9955 0.9955 1.0000

Prevalence 0.1668 0.1459 0.1341 0.1749 0.1869 0.1914

Detection Rate 0.1668 0.1459 0.1341 0.1712 0.1832 0.1914

Detection Prevalence 0.1668 0.1459 0.1341 0.1749 0.1869 0.1914

Balanced Accuracy 1.0000 1.0000 1.0000 0.9873 0.9879 1.0000

> linear.pred.test <- predict(best.linear, newdata = valid.df)

> confusionMatrix(linear.pred.test, valid.df$y)

Confusion Matrix and Statistics

Reference

Prediction 1 2 3 4 5 6

1 495 16 6 0 0 0

2 0 453 15 1 0 0

3 1 2 399 0 0 0

4 0 0 0 436 18 0

5 0 0 0 54 514 0

6 0 0 0 0 0 537

Overall Statistics

Accuracy : 0.9617

95% CI : (0.9541, 0.9683)

No Information Rate : 0.1822

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9539

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6

Sensitivity 0.9980 0.9618 0.9500 0.8880 0.9662 1.0000

Specificity 0.9910 0.9935 0.9988 0.9927 0.9776 1.0000

Pos Pred Value 0.9574 0.9659 0.9925 0.9604 0.9049 1.0000

Neg Pred Value 0.9996 0.9927 0.9917 0.9779 0.9924 1.0000

Prevalence 0.1683 0.1598 0.1425 0.1666 0.1805 0.1822

Detection Rate 0.1680 0.1537 0.1354 0.1479 0.1744 0.1822

Detection Prevalence 0.1754 0.1591 0.1364 0.1541 0.1927 0.1822

Balanced Accuracy 0.9945 0.9777 0.9744 0.9403 0.9719 1.0000

> poly.tune <- tune.svm(y ~ ., data=train.df,

+ kernel = "polynomial",

+ degree = c(3, 4, 5),

+ coef0 = c(0.1, 0.5, 1, 2, 3, 4))

> summary(poly.tune)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

degree coef0

5 2

- best performance: 0.01224268

- Detailed performance results:

degree coef0 error dispersion

1 3 0.1 0.02366755 0.006421952

2 4 0.1 0.03250850 0.008569012

3 5 0.1 0.04406999 0.010406605

4 3 0.5 0.01686668 0.006258424

5 4 0.5 0.01577862 0.004723210

6 5 0.5 0.01441900 0.004314694

7 3 1.0 0.01577899 0.004724439

8 4 1.0 0.01305901 0.004314770

9 5 1.0 0.01278708 0.004455429

10 3 2.0 0.01333056 0.003619084

11 4 2.0 0.01292240 0.004502840

12 5 2.0 0.01224268 0.004886450

13 3 3.0 0.01292240 0.004170597

14 4 3.0 0.01319488 0.005558118

15 5 3.0 0.01292277 0.005528303

16 3 4.0 0.01292258 0.005062233

17 4 4.0 0.01346661 0.004820624

18 5 4.0 0.01305864 0.005563530

> best.poly <- poly.tune$best.model

> poly.pred.train <- predict(best.poly, newdata = train.df)

> confusionMatrix(poly.pred.train, train.df$y)

Confusion Matrix and Statistics

Reference

Prediction 1 2 3 4 5 6

1 1226 0 0 0 0 0

2 0 1073 0 0 0 0

3 0 0 986 0 0 0

4 0 0 0 1286 0 0

5 0 0 0 0 1374 0

6 0 0 0 0 0 1407

Overall Statistics

Accuracy : 1

95% CI : (0.9995, 1)

No Information Rate : 0.1914

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 1

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6

Sensitivity 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

Specificity 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

Pos Pred Value 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

Neg Pred Value 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

Prevalence 0.1668 0.1459 0.1341 0.1749 0.1869 0.1914

Detection Rate 0.1668 0.1459 0.1341 0.1749 0.1869 0.1914

Detection Prevalence 0.1668 0.1459 0.1341 0.1749 0.1869 0.1914

Balanced Accuracy 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000

> poly.pred.test <- predict(best.poly, newdata = valid.df)

> confusionMatrix(poly.pred.test, valid.df$y)

Confusion Matrix and Statistics

Reference

Prediction 1 2 3 4 5 6

1 487 18 11 0 0 0

2 5 453 36 2 0 0

3 4 0 373 0 0 0

4 0 0 0 443 13 0

5 0 0 0 46 519 0

6 0 0 0 0 0 537

Overall Statistics

Accuracy : 0.9542

95% CI : (0.946, 0.9615)

No Information Rate : 0.1822

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9449

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6

Sensitivity 0.9819 0.9618 0.8881 0.9022 0.9756 1.0000

Specificity 0.9882 0.9826 0.9984 0.9947 0.9810 1.0000

Pos Pred Value 0.9438 0.9133 0.9894 0.9715 0.9186 1.0000

Neg Pred Value 0.9963 0.9927 0.9817 0.9807 0.9945 1.0000

Prevalence 0.1683 0.1598 0.1425 0.1666 0.1805 0.1822

Detection Rate 0.1653 0.1537 0.1266 0.1503 0.1761 0.1822

Detection Prevalence 0.1751 0.1683 0.1279 0.1547 0.1917 0.1822

Balanced Accuracy 0.9850 0.9722 0.9433 0.9485 0.9783 1.0000

> data.rf=randomForest(class\_train ~ ., data=X\_train, ntree=100, mtry=2, importance=TRUE)

> data.rf #Confustion matrix

Call:

randomForest(formula = class\_train ~ ., data = X\_train, ntree = 100, mtry = 2, importance = TRUE)

Type of random forest: classification

Number of trees: 100

No. of variables tried at each split: 2

OOB estimate of error rate: 4.82%

Confusion matrix:

1 2 3 4 5 6 class.error

1 1205 10 11 0 0 0 0.01712887

2 7 1051 15 0 0 0 0.02050326

3 5 33 948 0 0 0 0.03853955

4 0 1 0 1142 121 22 0.11197512

5 0 0 0 101 1269 4 0.07641921

6 0 2 1 9 12 1383 0.01705757

> varImpPlot(data.rf)

> #Predicting using random forest model

> X\_test$pred\_rf<-predict(object = data.rf,X\_test)

> #Checking the accuracy of the random forest model

> confusionMatrix(class\_test,X\_test$pred\_rf)

Confusion Matrix and Statistics

Reference

Prediction 1 2 3 4 5 6

1 475 2 19 0 0 0

2 18 447 6 0 0 0

3 37 59 324 0 0 0

4 0 1 0 378 103 9

5 0 0 0 21 511 0

6 0 0 0 1 7 529

Overall Statistics

Accuracy : 0.904

95% CI : (0.8928, 0.9144)

No Information Rate : 0.2107

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.8845

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6

Sensitivity 0.8962 0.8782 0.9284 0.9450 0.8229 0.9833

Specificity 0.9913 0.9902 0.9630 0.9556 0.9910 0.9967

Pos Pred Value 0.9577 0.9490 0.7714 0.7699 0.9605 0.9851

Neg Pred Value 0.9776 0.9750 0.9901 0.9910 0.9545 0.9963

Prevalence 0.1798 0.1727 0.1184 0.1357 0.2107 0.1826

Detection Rate 0.1612 0.1517 0.1099 0.1283 0.1734 0.1795

Detection Prevalence 0.1683 0.1598 0.1425 0.1666 0.1805 0.1822

Balanced Accuracy 0.9438 0.9342 0.9457 0.9503 0.9069 0.9900

> data.rf2=randomForest(class\_train ~ ., data=X\_train, ntree=200, mtry=2,do.trace=100, importance=TRUE)

ntree OOB 1 2 3 4 5 6

100: 4.33% 1.71% 1.86% 3.65% 11.04% 6.26% 0.92%

200: 3.50% 1.47% 1.12% 2.43% 9.72% 5.31% 0.36%

> data.rf2 #Confustion matrix

Call:

randomForest(formula = class\_train ~ ., data = X\_train, ntree = 200, mtry = 2, do.trace = 100, importance = TRUE)

Type of random forest: classification

Number of trees: 200

No. of variables tried at each split: 2

OOB estimate of error rate: 3.5%

Confusion matrix:

1 2 3 4 5 6 class.error

1 1208 8 10 0 0 0 0.01468189

2 1 1061 11 0 0 0 0.01118360

3 5 19 962 0 0 0 0.02434077

4 0 1 0 1161 110 14 0.09720062

5 0 0 0 72 1301 1 0.05312955

6 0 2 0 1 2 1402 0.00355366

> varImpPlot(data.rf2)

>

> #Predicting using random forest model

> X\_test$pred\_rf2<-predict(object = data.rf2,X\_test)

> X\_test$pred\_rf.prob2<-predict(object = data.rf2,X\_test,type="prob")

>

> #Checking the accuracy of the random forest model

> confusionMatrix(class\_test,X\_test$pred\_rf2)

Confusion Matrix and Statistics

Reference

Prediction 1 2 3 4 5 6

1 481 1 14 0 0 0

2 22 443 6 0 0 0

3 30 53 337 0 0 0

4 0 1 0 404 81 5

5 0 0 0 15 516 1

6 0 0 0 0 7 530

Overall Statistics

Accuracy : 0.9199

95% CI : (0.9095, 0.9295)

No Information Rate : 0.205

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9037

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6

Sensitivity 0.9024 0.8896 0.9440 0.9642 0.8543 0.9888

Specificity 0.9938 0.9886 0.9680 0.9656 0.9932 0.9971

Pos Pred Value 0.9698 0.9406 0.8024 0.8228 0.9699 0.9870

Neg Pred Value 0.9788 0.9778 0.9921 0.9939 0.9636 0.9975

Prevalence 0.1809 0.1690 0.1211 0.1422 0.2050 0.1819

Detection Rate 0.1632 0.1503 0.1144 0.1371 0.1751 0.1798

Detection Prevalence 0.1683 0.1598 0.1425 0.1666 0.1805 0.1822

Balanced Accuracy 0.9481 0.9391 0.9560 0.9649 0.9237 0.9930

> write\_rds(X\_train, "x\_train.rds")

> write\_rds(X\_test, "x\_test.rds")

> write\_rds(y.train, "y\_train.rds")

> write\_rds(y.test, "y\_test.rds")

> rm(X\_train)

> rm(X\_test)

> rm(y\_train)

> rm(y\_test)

> rm(y.train)

> rm(y.test)

> x\_train\_rds <- read\_rds("x\_train.rds")

> x\_test\_rds <- read\_rds("x\_test.rds")

> y\_train\_rds <- read\_rds("y\_train.rds")

> y\_test\_rds <- read\_rds("y\_test.rds")

>

> h2o.init(nthreads = -1)

Connection successful!

R is connected to the H2O cluster:

H2O cluster uptime: 5 minutes 49 seconds

H2O cluster timezone: America/New\_York

H2O data parsing timezone: UTC

H2O cluster version: 3.28.0.2

H2O cluster version age: 1 month and 2 days

H2O cluster name: H2O\_started\_from\_R\_haonguyen\_gig633

H2O cluster total nodes: 1

H2O cluster total memory: 1.56 GB

H2O cluster total cores: 4

H2O cluster allowed cores: 4

H2O cluster healthy: TRUE

H2O Connection ip: localhost

H2O Connection port: 54321

H2O Connection proxy: NA

H2O Internal Security: FALSE

H2O API Extensions: Amazon S3, XGBoost, Algos, AutoML, Core V3, TargetEncoder, Core V4

R Version: R version 3.6.2 (2019-12-12)

> data\_h2o <- as.h2o(

+ bind\_cols(y\_train\_rds, x\_train\_rds),

+ destination\_frame= "train.hex"

+ )

|======================================================================================================================| 100%

>

> new\_data\_h2o <- as.h2o(

+ bind\_cols(y\_test\_rds, x\_test\_rds),

+ destination\_frame= "test.hex"

+ )

|======================================================================================================================| 100%

>

> splits <- h2o.splitFrame(data = data\_h2o,

+ ratios = c(0.7, 0.15), # 70/15/15 split

+ seed = 1234

+ )

>

> train\_h2o <- splits[[1]] # from training data

> valid\_h2o <- splits[[2]] # from training data

> test\_h2o <- splits[[3]] # from training data

>

> y <- "y" # column name for outcome

> x <- setdiff(names(train\_h2o), y) # column names for predictors

> m1 <- h2o.deeplearning(

+ model\_id = "dl\_model\_first",

+ x = x,

+ y = y,

+ training\_frame = train\_h2o,

+ validation\_frame = valid\_h2o, ## validation dataset: used for scoring and

+ ## early stopping

+ #activation="Rectifier", ## default

+ #hidden=c(200,200), ## default: 2 hidden layers, 200 neurons each

+ epochs = 1 ## one pass over the training data

+ )

|======================================================================================================================| 100%

> summary(m1)

Model Details:

==============

H2ORegressionModel: deeplearning

Model Key: dl\_model\_first

Status of Neuron Layers: predicting y, regression, gaussian distribution, Quadratic loss, 152,801 weights/biases, 1.8 MB, 5,533 training samples, mini-batch size 1

layer units type dropout l1 l2 mean\_rate rate\_rms momentum mean\_weight weight\_rms mean\_bias bias\_rms

1 1 561 Input 0.00 % NA NA NA NA NA NA NA NA NA

2 2 200 Rectifier 0.00 % 0.000000 0.000000 0.013443 0.005550 0.000000 -0.000288 0.051570 0.477557 0.015290

3 3 200 Rectifier 0.00 % 0.000000 0.000000 0.018674 0.042560 0.000000 -0.005243 0.069083 0.992801 0.007436

4 4 1 Linear NA 0.000000 0.000000 0.000368 0.000125 0.000000 0.005961 0.078063 -0.003435 0.000000

H2ORegressionMetrics: deeplearning

\*\* Reported on training data. \*\*

\*\* Metrics reported on full training frame \*\*

MSE: 0.2900377

RMSE: 0.5385515

MAE: 0.4188668

RMSLE: 0.1408457

Mean Residual Deviance : 0.2900377

H2ORegressionMetrics: deeplearning

\*\* Reported on validation data. \*\*

\*\* Metrics reported on full validation frame \*\*

MSE: 0.3023441

RMSE: 0.5498583

MAE: 0.4306998

RMSLE: 0.1453166

Mean Residual Deviance : 0.3023441

Scoring History:

timestamp duration training\_speed epochs iterations samples training\_rmse training\_deviance training\_mae

1 2020-02-22 22:38:37 0.000 sec NA 0.00000 0 0.000000 NA NA NA

2 2020-02-22 22:38:39 5.401 sec 318 obs/sec 0.09643 1 497.000000 1.05271 1.10819 0.83992

3 2020-02-22 22:38:51 16.375 sec 484 obs/sec 1.07354 11 5533.000000 0.53855 0.29004 0.41887

training\_r2 validation\_rmse validation\_deviance validation\_mae validation\_r2

1 NA NA NA NA NA

2 0.63684 1.08088 1.16830 0.87226 0.61767

3 0.90495 0.54986 0.30234 0.43070 0.90106

Variable Importances: (Extract with `h2o.varimp`)

=================================================

Variable Importances:

variable relative\_importance scaled\_importance percentage

1 V97 1.000000 1.000000 0.001974

2 V89 0.997748 0.997748 0.001970

3 V57 0.991865 0.991865 0.001958

4 V81 0.990685 0.990685 0.001956

5 V224 0.988376 0.988376 0.001951

---

variable relative\_importance scaled\_importance percentage

556 V162 0.805387 0.805387 0.001590

557 V442 0.804502 0.804502 0.001588

558 V335 0.803958 0.803958 0.001587

559 V228 0.800956 0.800956 0.001581

560 V226 0.794395 0.794395 0.001568

561 V106 0.788546 0.788546 0.001557

> test\_prediction = h2o.predict(m1, newdata = as.h2o(test\_h2o[1]))

|======================================================================================================================| 100%

There were 50 or more warnings (use warnings() to see the first 50)

> h2o.performance(m1, newdata = as.h2o(test\_h2o))

H2ORegressionMetrics: deeplearning

MSE: 0.2891231

RMSE: 0.5377017

MAE: 0.4237559

RMSLE: 0.1383091

Mean Residual Deviance : 0.2891231