> 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\_h2o

y V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11

1 5 0.2791739 -0.02620065 -0.1232826 -0.9960915 -0.9834027 -0.9906751 -0.9970995 -0.9827498 -0.9893025 -0.9386916 -0.5761589

V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22

1 -0.8297115 0.843609 0.6824009 0.8378693 -0.9860933 -0.9999755 -0.999736 -0.9995037 -0.9971804 -0.9837991 -0.9860068

V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34

1 -0.6274463 -0.85093 -0.9118716 0.06143571 0.07483956 0.198204 -0.2643073 0.07254524 -0.1553198 0.323154 -0.170813 0.2949384

V35 V36 V37 V38 V39 V40 V41 V42 V43 V44 V45

1 -0.306081 0.4821478 -0.4701287 -0.305693 -0.3626541 0.5074589 0.9676152 -0.1439765 0.09985014 -0.9966456 -0.9813928

V46 V47 V48 V49 V50 V51 V52 V53 V54 V55 V56

1 -0.9784764 -0.9964569 -0.9809618 -0.9784558 0.8938171 -0.1637112 0.09342467 0.986821 -0.1213358 0.09575265 -0.4002781

V57 V58 V59 V60 V61 V62 V63 V64 V65 V66 V67 V68

1 0.9106207 -0.9693997 -0.9824196 -0.9959764 -0.9806634 -0.9797787 -1 -1 -0.5961008 -0.06493488 0.0754272 -0.08582296

V69 V70 V71 V72 V73 V74 V75 V76 V77 V78 V79 V80

1 0.09620771 -0.295036 0.2283097 -0.2062812 0.2048009 -0.3854102 0.3863727 -0.3871199 0.3852631 -0.9913087 -0.9688214 0.9842555

V81 V82 V83 V84 V85 V86 V87 V88 V89 V90 V91

1 0.07732061 0.02005764 -0.009864772 -0.9926974 -0.9875527 -0.9934976 -0.994266 -0.985717 -0.9914832 -0.987077 -0.9917862

V92 V93 V94 V95 V96 V97 V98 V99 V100 V101 V102

1 -0.9897689 0.9883902 0.9925438 0.9932176 -0.9928683 -0.9999236 -0.9998029 -0.9998829 -0.994678 -0.987033 -0.9888963

V103 V104 V105 V106 V107 V108 V109 V110 V111 V112 V113 V114

1 -0.8208042 -0.7549682 -0.8252786 0.122893 0.2764188 0.457445 0.1934143 0.1024047 -0.09910322 0.1946788 0.4842436 0.3576571

V115 V116 V117 V118 V119 V120 V121 V122 V123 V124 V125

1 -0.1870325 0.2980687 0.4518696 -0.1274952 -0.08327836 0.4570598 -0.04340998 -0.09138618 0.0855377 -0.9913848 -0.9924073

V126 V127 V128 V129 V130 V131 V132 V133 V134 V135 V136

1 -0.9875542 -0.9915889 -0.9931421 -0.9895849 -0.8853204 -0.9566563 -0.7432771 0.834164 0.9057528 0.8266338 -0.9822957

V137 V138 V139 V140 V141 V142 V143 V144 V145 V146 V147

1 -0.9997929 -0.9999023 -0.9998773 -0.9910313 -0.9941653 -0.994582 -0.9305511 -0.826618 -0.5434217 -0.1658855 -0.01288054

V148 V149 V150 V151 V152 V153 V154 V155 V156 V157 V158 V159

1 0.320055 -0.1651139 0.04455323 -0.1251933 0.07832123 0.1770276 0.1934017 -0.2071888 0.1124575 0.2020919 0.2101941 0.1411008

V160 V161 V162 V163 V164 V165 V166 V167 V168 V169 V170

1 -0.7253011 -0.09116989 -0.03633262 -0.06046466 -0.9911194 -0.996641 -0.9933289 -0.9912405 -0.9969579 -0.9940195 -0.9936763

V171 V172 V173 V174 V175 V176 V177 V178 V179 V180 V181

1 -0.9938002 -0.9889631 0.9892905 0.9981303 0.9941426 -0.9954961 -0.9999251 -0.9999862 -0.9999396 -0.9909086 -0.9970785

V182 V183 V184 V185 V186 V187 V188 V189 V190 V191 V192

1 -0.9954046 -0.5620311 -0.7313321 -0.6614345 0.009894985 -0.1375715 0.1259652 0.3161197 0.09433295 0.02617132 0.06966099

V193 V194 V195 V196 V197 V198 V199 V200 V201 V202 V203

1 0.2466532 0.2573553 -0.1368088 0.08731557 0.1490956 0.1966573 0.1404519 -0.3058977 -0.9865418 -0.9864213 -0.9864305

V204 V205 V206 V207 V208 V209 V210 V211 V212 V213 V214

1 -0.9864961 -0.9970448 -0.9865418 -0.9997374 -0.9835088 -0.5890063 -0.09285588 0.04639617 -0.0004664473 0.03714251 -0.9865418

V215 V216 V217 V218 V219 V220 V221 V222 V223 V224 V225

1 -0.9864213 -0.9864305 -0.9864961 -0.9970448 -0.9865418 -0.9997374 -0.9835088 -0.5890063 -0.09285588 0.04639617 -0.0004664473

V226 V227 V228 V229 V230 V231 V232 V233 V234 V235 V236

1 0.03714251 -0.993078 -0.9933808 -0.9931945 -0.9934017 -0.993012 -0.993078 -0.9998836 -0.9917361 -0.7923214 0.6616034

V237 V238 V239 V240 V241 V242 V243 V244 V245 V246 V247

1 -0.2474512 -0.2303149 -0.4364594 -0.9820599 -0.9873511 -0.9856316 -0.9900291 -0.9816856 -0.9820599 -0.9997679 -0.983966

V248 V249 V250 V251 V252 V253 V254 V255 V256 V257 V258

1 -0.2407192 -0.201985 0.05471192 0.1100716 -0.07942259 -0.9955022 -0.9952666 -0.9953048 -0.9953595 -0.9976518 -0.9955022

V259 V260 V261 V262 V263 V264 V265 V266 V267 V268 V269

1 -0.9999702 -0.9950011 -0.6830162 0.5953713 -0.2645688 -0.315723 -0.1638255 -0.9954906 -0.9835697 -0.9910798 -0.9963121

V270 V271 V272 V273 V274 V275 V276 V277 V278 V279 V280

1 -0.9832444 -0.9902291 -0.9945468 -0.9828241 -0.9890073 -0.9974035 -0.9872749 -0.9877543 -0.9944319 -0.9902589 -0.9965778

V281 V282 V283 V284 V285 V286 V287 V288 V289 V290 V291 V292 V293

1 -0.992812 -0.9999754 -0.9996969 -0.9998029 -0.9904425 -0.9919019 -0.9880605 -1 -0.8703979 -0.9441902 -1 -1 -1

V294 V295 V296 V297 V298 V299 V300 V301 V302 V303 V304

1 0.02904438 0.08030227 0.1856947 -0.599118 -0.9084493 -0.4609145 -0.8130567 -0.5668348 -0.7712461 -0.9999886 -0.9999767

V305 V306 V307 V308 V309 V310 V311 V312 V313 V314 V315

1 -0.9998343 -0.9998709 -0.9999924 -0.9999492 -0.9999645 -0.9999958 -0.9999864 -0.9998253 -0.9999912 -0.999975 -0.9999773

V316 V317 V318 V319 V320 V321 V322 V323 V324 V325 V326

1 -0.9999124 -0.9997191 -0.9997498 -0.9999442 -0.9999397 -0.9995535 -0.9998987 -0.9995122 -0.9998656 -0.9996727 -0.9999362

V327 V328 V329 V330 V331 V332 V333 V334 V335 V336 V337

1 -0.9996673 -0.9996458 -0.9996925 -0.9998733 -0.9998071 -0.9997748 -0.9998102 -0.9999293 -0.9998582 -0.9996655 -0.9996813

V338 V339 V340 V341 V342 V343 V344 V345 V346 V347 V348

1 -0.9999836 -0.9998044 -0.9998867 -0.9998438 -0.9997784 -0.9997918 -0.9999217 -0.9944466 -0.9887272 -0.9913542 -0.9913783

V349 V350 V351 V352 V353 V354 V355 V356 V357 V358 V359

1 -0.9869269 -0.9943908 -0.9894309 -0.987145 -0.9937904 -0.9934019 -0.9878742 -0.9942012 -0.997903 -0.9997671 -0.9653808

V360 V361 V362 V363 V364 V365 V366 V367 V368 V369 V370 V371 V372 V373

1 -0.9926573 -0.9999235 -0.9998031 -0.999883 -0.9917761 -0.9906849 -0.9932884 -1 -1 -1 -0.12 -0.56 -0.28 0.03579805

V374 V375 V376 V377 V378 V379 V380 V381 V382 V383 V384

1 -0.09303585 0.1680952 -0.2638514 -0.7572285 -0.3960393 -0.8296347 -0.5770384 -0.8933748 -0.9999978 -0.9999651 -0.9998432

V385 V386 V387 V388 V389 V390 V391 V392 V393 V394 V395

1 -0.9998654 -0.9999959 -0.9999304 -0.9999417 -0.9999992 -0.9999781 -0.9998269 -0.9999824 -0.9999426 -0.9999268 -0.9999009

V396 V397 V398 V399 V400 V401 V402 V403 V404 V405 V406

1 -0.999895 -0.9997965 -0.9998881 -0.9999061 -0.9996812 -0.9998463 -0.9996932 -0.9999999 -0.9997976 -0.9998832 -0.9997223

V407 V408 V409 V410 V411 V412 V413 V414 V415 V416 V417

1 -0.9997359 -0.9998056 -0.999856 -0.9998848 -0.9997237 -0.9998414 -0.9999425 -0.9998694 -0.9997353 -0.999204 -0.999662

V418 V419 V420 V421 V422 V423 V424 V425 V426 V427 V428

1 -0.999756 -0.9999203 -0.9998281 -0.9992066 -0.9998244 -0.9999237 -0.9871096 -0.9936015 -0.9871913 -0.9928104 -0.991646

V429 V430 V431 V432 V433 V434 V435 V436 V437 V438 V439

1 -0.9886776 -0.9896713 -0.9934607 -0.9865264 -0.9945184 -0.9918014 -0.9922807 -0.9897006 -0.9943438 -0.9931436 -0.9903443

V440 V441 V442 V443 V444 V445 V446 V447 V448 V449 V450 V451

1 -0.9999462 -0.9999514 -0.9998668 -0.9928778 -0.9962888 -0.9902236 -0.7236655 -0.8037541 -0.8172859 -1 -1 -0.7931035

V452 V453 V454 V455 V456 V457 V458 V459 V460 V461 V462

1 0.2168625 -0.1352454 -0.04972798 -0.5720876 -0.8736179 -0.1351185 -0.5422384 -0.3793527 -0.7565483 -0.9999637 -0.9998907

V463 V464 V465 V466 V467 V468 V469 V470 V471 V472 V473

1 -0.9999465 -0.9999733 -0.9998773 -0.9999031 -0.9998334 -0.9998929 -0.9999502 -0.9999481 -0.9998769 -0.9998597 -0.9999484

V474 V475 V476 V477 V478 V479 V480 V481 V482 V483 V484

1 -0.9999461 -0.9999305 -0.9999892 -0.9999924 -0.9999925 -0.9999859 -0.9999541 -0.9999898 -0.9999883 -0.999947 -0.9999908

V485 V486 V487 V488 V489 V490 V491 V492 V493 V494 V495

1 -0.9999796 -0.9999878 -0.9999478 -0.9999884 -0.9998773 -0.9999685 -0.9999566 -0.9999556 -0.9999493 -0.9998805 -0.9998793

V496 V497 V498 V499 V500 V501 V502 V503 V504 V505 V506

1 -0.9999495 -0.9998747 -0.9999398 -0.9999364 -0.9999101 -0.9998713 -0.9999516 -0.9875187 -0.986742 -0.9835237 -0.9902299

V507 V508 V509 V510 V511 V512 V513 V514 V515 V516 V517 V518

1 -0.998185 -0.9875187 -0.9997702 -0.9832153 -0.9070143 -1 0.0735815 -0.4684223 -0.7564936 -0.9927689 -0.9916998 -0.9890549

V519 V520 V521 V522 V523 V524 V525 V526 V527 V528 V529 V530

1 -0.9944546 -0.9955624 -0.9927689 -0.9998948 -0.9880552 -1 1 0.6789213 -0.7011307 -0.9096391 -0.9894128 -0.9878358

V531 V532 V533 V534 V535 V536 V537 V538 V539 V540 V541 V542

1 -0.9868499 -0.9867488 -0.9961994 -0.9894128 -0.9998756 -0.9891355 -0.7208908 -1 -0.0356842 -0.2300909 -0.511217 -0.9952207

V543 V544 V545 V546 V547 V548 V549 V550 V551 V552 V553

1 -0.9952369 -0.9957222 -0.9952731 -0.9957318 -0.9952207 -0.9999744 -0.995226 -0.9556959 -0.9365079 0.4045725 -0.1172902

V554 V555 V556 V557 V558 V559 V560 V561

1 -0.4828445 -0.03678797 -0.01289249 0.640011 -0.4853665 -0.8486494 0.1819348 -0.04766318

[ reached 'max' / getOption("max.print") -- omitted 5 rows ]

[1110 rows x 562 columns]

> test\_h2o[C(1)]

Error in C(1) : object not interpretable as a factor

> test\_h2o[C(0)]

Error in C(0) : object not interpretable as a factor

> test\_h2o[1]

y

1 5

2 5

3 5

4 5

5 5

6 5

[1110 rows x 1 column]

> 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