Machine Learning Hackathon challenge in Hackerearth platform

It is a classification problem. This dataset contains railway record of 2 month time span. The recorded information are like “Train Name”, “Station Name”, “Date_Time”, “Halt time on station”, “Destination Name” etc.

Objective: To predict the crowd of a train on a particular day, whether the train will have low or medium or high crowd. Keeping different factors in mind on this dataset different insight can also be found out.

Target : Target variable is named as “target” with three levels high, medium and low.

Required Packages:-

```
library(randomForest) #Random Forest Model
library(caret)        # Data partition and for accuaracy check
library(lubridate)    # Date manipulation
library(xgboost)      # Extreme Gradient Boosting
library(Matrix)       # Matrix maniplulation
library(dplyr)        # Easier way to handle data
library(geosphere)    # Geograpic Distance calculator
library(e1071)        # Support Vector Machine
library(MLmetrics)    # Accuracy Matrix Calculation
library(purrr)        # Eval and mapping
```

About data

```
setwd("C:\\Users\\Sanjeeb\\Desktop\\Study\\Hitachi Hackerearth\\DataSet")
DF <- read.csv("Train.csv")
head(DF,3)
```

```
##           id_code current_date current_time source_name destination_name
## 1 isfywypmkqqhyft   2016-07-27  08:05:51 PM station$147        station$1
## 2 mqsfxyvuqpbwomk   2016-07-27  08:06:11 PM station$147        station$1
## 3 alspwwtbdvqsgby   2016-07-27  08:08:57 PM station$147        station$1
##   train_name target country_code_source longitude_source latitude_source
## 1     ICZVZS   high               whber         4.356801        50.84566
## 2     ICZVZS   high               whber         4.356801        50.84566
## 3     ICZVZS   high               whber         4.356801        50.84566
##   mean_halt_times_source country_code_destination longitude_destination
## 1               634.1647                                            NA
## 2               634.1647                                            NA
## 3               634.1647                                            NA
##   latitude_destination mean_halt_times_destination current_year
## 1                   NA                          NA         2016
## 2                   NA                          NA         2016
## 3                   NA                          NA         2016
##   current_week current_day is_weekend
## 1           30   Wednesday      False
## 2           30   Wednesday      False
## 3           30   Wednesday      False
```

Missing Value Treatment

```
#Missing Value Counts
sapply(DF,function(x) sum(is.na(x)))
```

```
##                     id_code                current_date
##                           0                           0
##                current_time                 source_name
##                           0                           0
##            destination_name                  train_name
##                           0                           0
##                      target         country_code_source
##                           0                           0
##            longitude_source             latitude_source
##                           1                           1
##      mean_halt_times_source    country_code_destination
##                           1                           0
##       longitude_destination        latitude_destination
##                          33                          33
## mean_halt_times_destination                current_year
##                          33                           0
##                current_week                 current_day
##                           0                           0
##                  is_weekend
##                           0
```

Only 33 rows have missing value (NA’s) and it is missing for most of the features so removed those rows which is really very less informative.

```
Data <- DF[complete.cases(DF),]
```

Feature Engineering

In feature engineering, I have derived 3 new features which are most important and relevent for train crowd prediction and those are-
1. Geographical distance of one station to another station.

2. Day count of WeeK (example: Sunday is counted as 1, Monday is counted as 2…)
Day count of Month (example: 5th day of Month is counted as 5, 12th day of Month is counted as 12…) and
similarly Day count of Year (example: 5th Jan is counted as 5, 5th Feb is counted as 37 …)

3. Time duration for a train to reach one station to another station.

1. Calculating the geographic distance based on longtitude and latitude

```
get_geo_distance  <-   function(long1, lat1, long2, lat2) {
  longlat1  <-   map2(long1, lat1, function(x,y) c(x,y))
  longlat2  <-   map2(long2, lat2, function(x,y) c(x,y))
  distance_list  <-   map2(longlat1, longlat2, function(x,y) geosphere::distHaversine(x, y))
  distance  <-   unlist(distance_list) / 1000
  distance
}
Data$Distance <-  get_geo_distance(Data$longitude_source,Data$latitude_source,Data$longitude_destination,Data$lat
itude_destination)
head(Data$Distance)
```

```
## [1] 19.556665  0.000000 81.217882  9.448971 50.025797  5.578035
```

2. Calculating the days count for the Week, Month and Year

```
Data$W_Day <-   wday(as.Date(Data$current_date) )
Data$M_Day <-   mday(as.Date(Data$current_date) )
Data$Y_Day <-   yday(as.Date(Data$current_date) )
head(Data$W_Day)
```

```
## [1] 4 5 5 5 5 5
```

```
head(Data$M_Day)
```

```
## [1] 27 28 28 28 28 28
```

```
head(Data$Y_Day)
```

```
## [1] 209 210 210 210 210 210
```

3. Calculating the time duration for a train to reach one station to another station

```
STD_Time <-   strptime( Data$current_time, "%I:%M:%S %p")
Data$current_time <-  hour(STD_Time)*60+minute(STD_Time) # converted the time into minute
Data <-  Data %>% group_by(train_name) %>%arrange(Y_Day,current_time) %>% mutate(Time_Diff =   ( (lead(Y_Day)-Y_D
ay)*24*60) +(lead(current_time)-current_time) )
Data <-  as.data.frame(Data)
Data$Time_Diff[is.na(Data$Time_Diff)] <-  median(Data$Time_Diff,na.rm=TRUE)
head(Data$Time_Diff)
```

```
## [1]    12.5    14.0 75503.0 38751.0 21646.0    12.5
```

Removed redundent features which is not required. They are mostly zero variance or near zero variance.

```
Data$id_code <-  NULL
Data$current_date <-  NULL
Data$train_name  <-  NULL
Data$country_code_source <-  NULL
Data$country_code_destination <-  NULL
Data$current_year <-  NULL
Data$source_name <-  NULL
Data$destination_name <-  NULL
#Now data looks
head(Data,3)
```

```
##   current_time target longitude_source latitude_source
## 1         1421   high         4.360846        50.85966
## 2            4    low         5.497685        50.96706
## 3           18    low         5.497685        50.96706
##   mean_halt_times_source longitude_destination latitude_destination
## 1              640.26590              4.482785             51.01765
## 2               39.47688              5.497685             50.96706
## 3               39.47688              4.356801             50.84566
##   mean_halt_times_destination current_week current_day is_weekend Distance
## 1                   306.52312           30   Wednesday      False 19.55666
## 2                    39.47688           30    Thursday      False  0.00000
## 3                   634.16474           30    Thursday      False 81.21788
##   W_Day M_Day Y_Day Time_Diff
## 1     4    27   209      12.5
## 2     5    28   210      14.0
## 3     5    28   210   75503.0
```

Data Partition

Splitting data into 2 parts, 70% as Train Data and 30% as Test Data

```
T_index  <-  createDataPartition(Data$target,p = 0.7, list = FALSE)
Train_Data  <-  Data[T_index,]
Test_Data  <-  Data[-T_index,]
```

Model Building

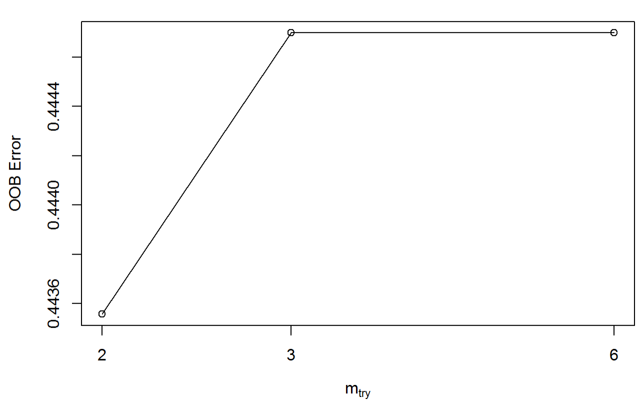
Since it’s a classification problem and the features are non linear, so decided to use 3 Machine Learning Models here namely Random Forest, Support Vector Machine and Extreme Gradient Boosting Model.

Accuracy metric is F1 score which is given in the challenge to measure the performance of a model.

1. Random Forest Model

```
# Parameter tuning for random forest model. 2 is column position for target variable.
Tune_RF <- tuneRF(Train_Data[,-2],Train_Data[,2], doBest = TRUE)
```

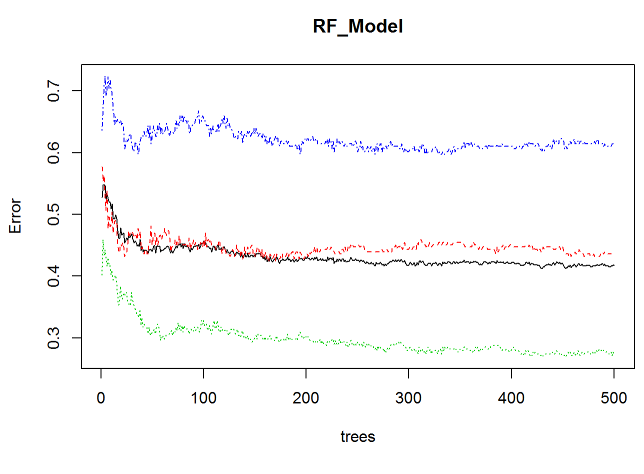
```
## mtry = 3  OOB error = 44.47%
## Searching left ...
## mtry = 2     OOB error = 44.36%
## 0.002564103 0.05
## Searching right ...
## mtry = 6     OOB error = 44.47%
## 0 0.05
```




```
# Find lowest  OOB % and give best no. of features for trees i.e. mtry
Tune_RF$mtry
```

```
## [1] 2
```

```
RF_Model <-  randomForest(target~., data = Train_Data, mtry =Tune_RF$mtry)
# To decide no.of trees required for the Model i.e. ntree
plot(RF_Model)
```




```
# Error is almost statble after ntree 100, so considered 100
RF_Model <-  randomForest(target~., data = Train_Data, ntree = 100, mtry =Tune_RF$mtry, nodesize = 100)

# Prediction for Train data
RF_Pred  <-  predict(RF_Model, Train_Data[,-2])
table(Train_Data$target,RF_Pred)
```

```
##         RF_Pred
##          high low medium
##   high    179  68     17
##   low      58 302     12
##   medium   69 121     51
```

```
# Prediction for Test data
RF_Pred1  <-  predict(RF_Model, Test_Data[,-2])
table(Test_Data$target,RF_Pred1)
```

```
##         RF_Pred1
##          high low medium
##   high     74  30      8
##   low      33 118      8
##   medium   44  50      8
```

Accuracy checking for Random Forest : F1 Score

```
# Train Accuracy
F1_Score(Train_Data$target,RF_Pred, positive = NULL)
```

```
## [1] 0.6280702
```

```
# Test Accuracy
 F1_Score(Test_Data$target,RF_Pred1, positive = NULL)
```

```
## [1] 0.5627376
```

2. Support Vector Machine

```
# Parameter tuning for SVM model. 2 is column position for target variable.
Tune_SVM <-tune(svm,target~., data = Train_Data, ranges = list(epsilon=seq(0,1,0.2), cost=2^(1:5)) )
Tune_SVM$best.parameters
```

```
##    epsilon cost
## 25       0   32
```

```
SVM_Model <-  svm(target~., data = Train_Data, cost =Tune_SVM$best.parameters$cost,
              epsilon = Tune_SVM$best.parameters$epsilon)
# Prediction for Train data
SV_Pred  <-  predict(SVM_Model, Train_Data[,-2])
table(Train_Data$target,SV_Pred)
```

```
##         SV_Pred
##          high low medium
##   high    226  16     22
##   low      23 329     20
##   medium   24  25    192
```

```
# Prediction for Test data
SV_Pred1  <-  predict(SVM_Model, Test_Data[,-2])
table(Test_Data$target,SV_Pred1)
```

```
##         SV_Pred1
##          high low medium
##   high     62  23     27
##   low      36  92     31
##   medium   38  29     35
```

Accuracy checking for Support Vetor Machine : F1 Score

```
# Train Accuracy
F1_Score(Train_Data$target,SV_Pred, positive = NULL)
```

```
## [1] 0.8417132
```

```
# Test Accuracy
F1_Score(Test_Data$target,SV_Pred1, positive = NULL)
```

```
## [1] 0.5
```

3. Extreme Gradient Boosting

```
# Create matrix - One-Hot Encoding for Factor variables
XGB_Train <-  Train_Data
XGB_Test <-  Test_Data
XGB_Train$target <- ifelse(XGB_Train$target=="low",0, ifelse(XGB_Train$target=="medium",1,2) )
XGB_Test$target <- ifelse(XGB_Test$target=="low",0, ifelse(XGB_Test$target=="medium",1,2) )

# Model matrix for Train
trainm <- sparse.model.matrix(target ~ .-1, data = XGB_Train)
train_label <- XGB_Train[,"target"]
train_matrix <- xgb.DMatrix(data = as.matrix(trainm), label = train_label)

# Model matrix for Test
testm <- sparse.model.matrix(target~.-1, data = XGB_Test)
test_label <- XGB_Test[,"target"]
test_matrix <- xgb.DMatrix(data = as.matrix(testm), label = test_label)

# Objective Parameters
nc <- length(unique(train_label))
xgb_params <- list("objective" = "multi:softmax",
                   "eval_metric" = "mlogloss",
                   "num_class" = nc)
watchlist <- list(train = train_matrix, test = test_matrix)
# Extreme Gradient Boosting Model
# parameters are tuned manually to save computational time
XGB_Model <- xgb.train(params = xgb_params,
                       data = train_matrix,
                       nrounds = 300,
                       watchlist = watchlist,
                       eta = 0.01,
                       max.depth = 3,
                       verbose=0
                      )

# Prediction for Train data
XGB_Pred  <-  predict(XGB_Model, as.matrix(trainm) , type = "raw")
XGB_Pred <- ifelse(XGB_Pred==0,"low", ifelse(XGB_Pred==1,"medium","high") )
table(Train_Data$target,XGB_Pred)
```

```
##         XGB_Pred
##          high low medium
##   high    198  37     29
##   low      56 301     15
##   medium   54  73    114
```

```
# Prediction for Test data
XGB_Pred1  <-  predict(XGB_Model, as.matrix(testm), type = "raw")
XGB_Pred1 <- ifelse(XGB_Pred1==0,"low", ifelse(XGB_Pred1==1,"medium","high") )
table(Test_Data$target,XGB_Pred1)
```

```
##         XGB_Pred1
##          high low medium
##   high     66  21     25
##   low      39 102     18
##   medium   34  33     35
```

Accuracy checking for XGB : F1 Score

```
# Train Accuracy
F1_Score(Train_Data$target,XGB_Pred, positive = NULL)
```

```
## [1] 0.6923077
```

```
# Test Accuracy
F1_Score(Test_Data$target, XGB_Pred1, positive = NULL)
```

```
## [1] 0.5258964
```

We observed that:-

All 3 models in the above are perfoming average in the dataset (produced average accuracy each time i.e. 50%–55% for Test data). Train accuracy and Test accurcay is not so close because of very less data (data has 1.2k row only) and high variation exist in data.

So I am going to consider a final model (stacking model) based on the result of above 3 models with the following rules:-
1. If all 3 model gives same result, considered that result for final
2. If any 2 model gives same result, considered that result for final
3. If all 3 model gives different result, considered RF model result for final because RF model has highest accuracy.

Final Model (Stacking Model)

```
# Predicting for RF model
target1  <-  predict(RF_Model, Test_Data[,-2])
# Predicting for SVM model
target2 <-  predict(SVM_Model, Test_Data[,-2])
# Predicting for XGB model
XGB_Data<- sparse.model.matrix( ~.-1, data = Test_Data[,-2])
target3 <-  predict(XGB_Model, as.matrix(XGB_Data) )
target3 <-ifelse(target3==0,"low", ifelse(target3==1,"medium","high") )

#Combining all 3 models output
Target_Data=cbind.data.frame(A=target1, B=target2, C=target3)

getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

Test_target=apply(Target_Data,1,getmode)
table(Test_target)
```

```
## Test_target
##   high    low medium
##    145    175     53
```

Final accuracy from the Stacking Model : F1 Score

```
# Final Accuracy
F1_Score(Test_Data$target, Test_target, positive = NULL)
```

```
## [1] 0.5836576
```

The final accuracy is comparatively better than the other 3 models.

Thanks a lot

(Sanjeeb)