Project: Building predictive model for Prediction of probability of Store being opened using Predictive Analytics in R.

Problem Statement: Part 1

This data set is related with retail domain and challenge is to predict whether a store should get opened or not based on certain factors such as sales, population,area etc.

We have given you two datasets , store_train.csv and store_test.csv . You need to use data store_train to build predictive model for response variable 'store'. store_test data contains all other factors except 'store', you need to predict that using the model that you developed and submit your predicted values in a csv files.

You must submit the probability scores, not the hard classes.
If you are using decision trees or random forest here, probability scores can be calculated as
score=predict(rf_model,newdata= testdata, type="prob")[,2]
score=predict(tree_model,newdata= testdata, type='vector')[,2]

Note : you need to submit probability of outcome being 1

Evaluation Criterion : auc score on test data. larger auc score, better Model

Solution:

```
setwd("E:/R/retail")
#Next we need to import both train & test data sets.
s_train=read.csv("store_train.csv",stringsAsFactors = F)
s_test=read.csv("store_test.csv",stringsAsFactors = F)

#Let us load data wrangling library dplyr so as to glimpse our data.
library(dplyr)
glimpse(s_train)
glimpse(s_test)

#** Understanding our Data **
#Each row represnts characteristic of a single planned store.We can see from above that many categorical data has been coded to mask the data.Here is the interpretation for the columns Id : store id numeric sale figures for 5 types : sales0 sales1 sales2 sales3 sales4
#country : categorical :: coded values for country
#State : categorical :: coded values for State
#CouSub : numeric ::subscription values at county level
#countyname : Categorical ::county names
#storecode : categorical :: store codes
#Areaname : categorical :: name of the area , many times it matches with county name
#countytownname : categorical :: county town name
#population : numeric :: population of the store area
#state_alpha : categorical :: short codes for state
#store_Type : categorical :: type of store
#store : categorical 1/0 : target indicator var 1=opened 0=not opened
```

#** Data Preparation **
#We'll combine our two datasets so that we do not need to prepare data separately for them. And we'll also avoid problem of dealing with different columns in different datasets. However before combining them, we'll need to add response column to test because number of columns need to be same for two datasets to stack vertically.We are also going to add an identifier column 'data' which will recognize whether it is from train or test.

```
s_test$store=NA
s_train$data="train"
s_test$data="test"
s=rbind(s_train,s_test)
```

#Let us glimpse our combined data sets s using glimpse & str function.

```
glimpse(s)
str(s)
```

#Many categorical data like 'country' & 'State' has already been coded to mask the data. We can see the same using frequency table as shown below.

```
table(s$country)
table(s$State)
```

#Since we will be using random forest we need to convert data type of response (which is store in this case) to factor type using function as.factor. This is how randomforest differentiates from regression & classification.If we need to build a regression model then response variable should be kept numeric else factor for classification.

```
s$store=as.factor(s$store)
```

#Let us see if store column has been changed to factor type or not using glimpse function again from dplyr package.

```
glimpse(s)
```

#Next we will convert all categorical variables to dummies. We will write a function which will take care of that instead of converting them one by one.

```
CreateDummies=function(data,var,freq_cutoff=0){
 t=table(data[,var])
 t=t[t>freq_cutoff]
 t=sort(t)
 categories=names(t)[-1]
 for( cat in categories){
   name=paste(var,cat,sep="_")
   name=gsub(" ","",name)
   name=gsub("-","_",name)
   name=gsub("\\?","Q",name)
   name=gsub("<","LT_",name)
   name=gsub("\\+","",name)
   name=gsub("\\/","_",name)
   name=gsub(">","GT_",name)
   name=gsub("=","EQ_",name)
   name=gsub(",","",name)
```

```
   data[,name]=as.numeric(data[,var]==cat)
  }
  data[,var]=NULL
  return(data)
}

#Let us have a look at our categorical variables by writing following lines of codes
names(s)[sapply(s,function(x) is.character(x))]

#Now we will check for High-Cardinality in the categorical variables i.e we will check for variables with many distinct values. We will discard those variables from our modelling. Because including these attributes by standard dummy encoding increases the dimensionality of the data to such an extent that either the classification technique is unable to process them or if one would use some regularized linear technique that is able to cope with huge dimensions, it leads to a model with thousands or even millions of features, thereby losing the often required comprehensibility aspect.
length(unique(s$countyname))
length(unique(s$storecode))
length(unique(s$Areaname))
length(unique(s$countytownname))
length(unique(s$state_alpha))
length(unique(s$store_Type))

#We will ignore columns or variables like countyname,storecode,Areaname,countytownname for their High-Cardinality. Further we will ignore data column for obvious reason.
s=s %>% select(-countyname,-storecode,-Areaname,-countytownname)

#Above codes will discard those four variables & we are left with 14 variables now. Next Let us make dummies for the rest of columns - state_alpha & store_Type.
cat_cols=c("state_alpha","store_Type")

for(cat in cat_cols){
  s=CreateDummies(s,cat,100)
}
#This will increase our columns to 26 as we can glimpse the same as shown below.
glimpse(s)

#Let us see if there is any missing values in our data. We will use lapply function which will give output in list format as shown below.
lapply(s,function(x) sum(is.na(x)))

#From above we can see that We do have missing values in columns like country, population & store. Next we impute those missing values with the mean of train data as shown below.
for(col in names(s)){
  if(sum(is.na(s[,col]))>0 & !(col %in% c("data","store"))){
    s[is.na(s[,col]),col]=mean(s[s$data=='train',col],na.rm=T)
  }
}
```

```
#We can always cross check if those NAs has been replaced with mean or not by using lapply function again.
lapply(s,function(x) sum(is.na(x)))

#Now we are done with data preparation , lets separate the data next.
s_train=s %>% filter(data=="train") %>% select(-data)
s_test=s %>% filter(data=="test") %>% select(-data,-store)

#Next we will break our train data into 2 parts. We will build model on one part & check its performance on the other.
set.seed(2)
s=sample(1:nrow(s_train),0.8*nrow(s_train))
s_train1=s_train[s,]
s_train2=s_train[-s,]

#** Model Building **
#Let us load the package randomForest first
library(randomForest)

#Next we will build our model with 5 variables randomly subsetted at each node i.e mtry & let just say we want to grow 100 such trees.
model_rf=randomForest(store~.-Id,data=s_train1,mtry=5,ntree=100)

#Let us see what does this model_rf represent
model_rf

#** Model Validation **
#Lets see performance of this model on the validation data s_train2 that we kept aside.
val.score=predict(model_rf,newdata=s_train2,type='response')

#Again we need to check the accuracy using confusionMatrix from caret package. What we will get is an accuracy of 78% which seems to be a fair model.
library(caret)
confusionMatrix(val.score,s_train2$store)

#Now let us calculate probability score for our validation data set s_train2.
val.prob_score=predict(model_rf,newdata=s_train2,type='prob')

#In order to check the performance of our model let us calculate its auc score. For that we need to first import a package named ‘pROC’.
library(pROC)
auc_score=auc(roc(s_train2$store,val.prob_score[,1]))
auc_score

#We can also plot our auc
plot(roc(s_train2$store,val.prob_score[,1]))
```

#Next we will build the random forest model on the entire training data set ‘s_train’ & predict the same on test data set ‘s_test’

```
model_rf_final=randomForest(store~.-Id,data=s_train,mtry=5,ntree=100)
model_rf_final
```

#We will now use this model to predict probability score for test data .

```
test.score=predict(model_rf_final,newdata = s_test,type='prob')[,1]
test.score
```

#** Variable Importance **

#We will run below codes to find out the importance of variable. Higher the mean decrease ginni for any variable better is the variable for prediction. So population is the most important variable.

```
d=importance(model_rf_final)
d=as.data.frame(d)
d$VariableNames=rownames(d)
d %>% arrange(desc(MeanDecreaseGini))
varImpPlot(model_rf_final)
```

Save the scores to a csv file

```
write.csv(score, file = "final Store prediction.csv", row.names = FALSE)
```