R Code

Santander Customer Transaction Prediction

```
rm(list=ls(all=T))

#Loading Libraries:-
library(tidyverse)
library(moments)
library (DataExplorer)
library(lattice)
library(ggplot2)
library(caret)
library (Matrix)
#library(pdp)
library(mlbench)
library(caTools)
library(randomForest)
library(glmnet)
library(mlr)
library(vita)
library(rBayesianOptimization)
library(lightgbm)
library(stats)
library(pROC)
library(grid)
library(zoo)
library (DMwR)
library (ROSE)
library(caret)
library(yardstick)

#Setting Directory:-
Setwd ("C:/Users/Poo/Documents/edWisor/Project 02")

########--Importing the training Data-- #############
df_train=read.csv("train.csv")

#Dimension of the train data:-
dim(df_train)

#Summary of the train dataset:-
str(df_train)

########-- Importing the test data--- #########
df_test=read.csv("test.csv")
head(df_test)

#Dimension of test dataset: -
dim(df_test)
```

```r
#Typecasting the target variable: -
df_train$target=as.factor(df_train$target)

#Target class count in train data:-
table(df_train$target)

#Percentage count of target class in train data:-
table(df_train$target)/length(df_train$target)*100

#Bar plot for count of target classes in train data:-
plot1=ggplot(df_train,aes(target))+theme_bw()+geom_bar(stat='count',fill='green')

#Violin with jitter plots for target classes
plot2=ggplot(df_train,aes(x=target,y=1:nrow(df_train)))+theme_bw()+geom_violin(fill='magenta')+
  facet_grid(df_train$target)+geom_jitter(width=0.02)+labs(y='Index')
print(plot2)

########## Distribution on train data############
#Distribution of train attributes from 3 to 102:-
for (var in names(df_train)[c(3:102)]){
  target<-df_train$target
  plot<-ggplot (df_train, aes(df_train[[var]],fill=target)) +
    geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()
  print(plot)
}
plot3=plot
#Distribution of train attributes from 103 to 202:-
for (var in names(df_train)[c(103:202)]){
  target<-df_train$target
  plot4<-ggplot(df_train, aes(df_train[[var]],fill=target)) +
    geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()
  print(plot4)
}
############ Distribution on test data###########
#Distribution of test attributes from 2 to 101:-
plot5=plot_density(df_test[,c(2:101)],ggtheme = theme_classic(),geom_density_args =
list(color='red'))
print(plot5)
#Distribution of test attributes from 102 to 201:-
plot6=plot_density(df_test[,c(102:201)],ggtheme = theme_classic(),geom_density_args =
list(color='red'))
print(plot6)
###########################  Missing Value Analysis  ####################################
#Finding the missing values in train data
missing_val<-data.frame(missing_val=apply(df_train,2,function(x){sum(is.na(x))}))
missing_val<-sum(missing_val)
missing_val

#Finding the missing values in test data
missing_val<-data.frame(missing_val=apply(df_test,2,function(x){sum(is.na(x))}))
missing_val<-sum(missing_val)
```

```
missing_val

##########################  Feature Scaling ##################################3
#Applying the function to find skewness values per row in train and test data.
train_skew<-apply(df_train[,-c(1,2)],MARGIN=1,FUN=skewness)
test_skew<-apply(df_test[,-c(1)],MARGIN=1,FUN=skewness)
ggplot()+
  #Distribution of skewness values per row in train data

geom_density(aes(x=train_skew),kernel='gaussian',show.legend=TRUE,color='green')+theme_classic
()+
  #Distribution of skewness values per column in test data
  geom_density(aes(x=test_skew),kernel='gaussian',show.legend=TRUE,color='blue')+
  labs(x='skewness values per row',title="Distribution of skewness values per row in train and test
dataset")

#Applying the function to find skewness values per column in train and test data.
train_skew<-apply(df_train[,-c(1,2)],MARGIN=2,FUN=skewness)
test_skew<-apply(df_test[,-c(1)],MARGIN=2,FUN=skewness)
ggplot()+
  #Distribution of skewness values per column in train data

geom_density(aes(x=train_skew),kernel='gaussian',show.legend=TRUE,color='green')+theme_classic
()+
  #Distribution of skewness values per column in test data
  geom_density(aes(x=test_skew),kernel='gaussian',show.legend=TRUE,color='blue')+
  labs(x='skewness values per column',title="Distribution of skewness values per column in train and
test dataset")
#Correlations in train data:-
#convert factor to int
df_train$target<-as.numeric(df_train$target)
train_correlation<-cor(df_train[,c(2:202)])
train_correlation
#check correlation among top 10 observations
library(ggcorrplot)
ggcorrplot(train_correlation[1:10,1:10])

#Observation:- We can observe that correlation between train attributes is very small.

#Correlations in test data
test_correlation<-cor(df_test[,c(2:201)])
test_correlation
ggcorrplot(test_correlation[1:10,1:10])

#Observation:- We can observe that correlation between test attributes is very small.

###########################  Feature Engineering  ##################################

#Variable Importance:-Variable importance is used to see top features in dataset based on mean
decreases gini .
#Building a simple model to find features which are imp:-
```

```
#Split the training data using simple random sampling
train_index<-sample(1:nrow(df_train),0.75*nrow(df_train))
#train data
train_data<-df_train[train_index,]
#validation data
valid_data<-df_train[-train_index,]
#dimension of train and validation data
dim(train_data)
dim(valid_data)

#Random forest classifier:-
#Training the Random forest classifier
set.seed(2732)
#convert to int to factor
train_data$target<-as.factor(train_data$target)
#setting the mtry
mtry<-floor(sqrt(200))
#setting the tunegrid
tuneGrid<-expand.grid(.mtry=mtry)
#fitting the ranndom forest
rf<-randomForest(target~.,train_data[,-c(1)],mtry=mtry,ntree=10,importance=TRUE)

#Feature importance by random forest-
#Variable importance
VariableImp<-importance(rf,type=2)
VariableImp
#Handling of imbalanced data- Now we are going to explore 5 different approaches for dealing with
imbalanced datasets.
#Change the performance metric
#Oversample minority class
#Undersample majority class
#ROSE
#LightGBM

#Logistic Regression Model:-
#Split the data using simple random sampling:-
set.seed(689)
train.index<-sample(1:nrow(df_train),0.8*nrow(df_train))
#train data
train.data01<-df_train[train.index,]
#validation data
valid.data01<-df_train[-train.index,]
#dimension of train data
dim(train.data01)
#dimension of validation data
dim(valid.data01)
#target classes in train data
table(train.data01$target)
#target classes in validation data
table(valid.data01$target)
```

```r
#Training and validation dataset

#Training dataset
X1_t<-as.matrix(train.data01[,-c(1,2)])
y1_t<-as.matrix(train.data01$target)
#validation dataset
X2_v<-as.matrix(valid.data01[,-c(1,2)])
y2_v<-as.matrix(valid.data01$target)
#test dataset
test<-as.matrix(df_test[,-c(1)])

#Logistic regression model
set.seed(667) # to reproduce results
lr<-glmnet(X1_t,y1_t, family = "binomial")
summary(lr)

#Cross validation prediction
set.seed(8909)
cv_lr <- cv.glmnet(X1_t,y1_t,family = "binomial", type.measure = "class")

cv_lr

#Plotting the missclassification error vs log(lambda) where lambda is regularization parameter
#Minimum lambda
cv_lr$lambda.min
#plot the auc score vs log(lambda)
plot(cv_lr)

#Observation:-We can observed that miss classification error increases as increasing the
log(Lambda).

#Model performance on validation dataset
set.seed(5363)
cv_predict.lr<-predict(cv_lr,X2_v,s = "lambda.min", type = "class")
cv_predict.lr

#Observation:-Accuracy of the model is not the best metric to use when evaluating the imbalanced
datasets as it may be misleading. So, we are going to change the performance metric.

#Confusion Matrix:-
set.seed(689)
#actual target variable
target<-valid.data01$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.lr<-as.factor(cv_predict.lr)
confusionMatrix(data=cv_predict.lr,reference=target)
```

```
#Reciever operating characteristics(ROC)-Area under curve(AUC) score and curve:-
#ROC_AUC score and curve
set.seed(892)
cv_predict.lr<-as.numeric(cv_predict.lr)
A=valid.data01
B=as.numeric(cv_predict.lr)
C=X2_v
A=cbind(A,B)
C=cbind(C,B)
roc(data=C,response=target,predictor=B,auc=TRUE,plot=TRUE)

 #Oversample Minority Class:-
#-Adding more copies of minority class.
#-It cab be a good option we dont have that much large data to work.
#-Drawback of this process is we are adding info. That can lead to overfitting or poor performance
on test data.
#Undersample Mojorityclass:-
#-Removing some copies of majority class.
#-It can be a good option if we have very large amount of data say in millions to work.
#-Drawback of this process is we are removing some valuable info. that can leads to underfitting &
poor performance on test data.

#Both Oversampling and undersampling techniques have some drawbacks. So, we are not going to
use this models for this problem and also we will use other best algorithms.

#Random Oversampling Examples(ROSE)- It creates a sample of synthetic data by enlarging the
features space of minority and majority class examples.

#Random Oversampling Examples(ROSE)
set.seed(699)
train.rose <- ROSE(target~., data =train.data01[,-c(1)],seed=32)$data
#target classes in balanced train data
table(train.rose$target)
valid.rose <- ROSE(target~., data =valid.data01[,-c(1)],seed=42)$data
#target classes in balanced valid data
table(valid.rose$target)

#Logistic regression model
set.seed(462)
lr_rose <-glmnet(as.matrix(train.rose),as.matrix(train.rose$target), family = "binomial")
summary(lr_rose)

#Cross validation prediction
set.seed(473)
cv_rose = cv.glmnet(as.matrix(valid.rose),as.matrix(valid.rose$target),family = "binomial",
type.measure = "class")
cv_rose

#Plotting the missclassification error vs log(lambda) where lambda is regularization parameter:-
#Minimum lambda
cv_rose$lambda.min
```

```
#plot the auc score vs log(lambda)
plot(cv_rose)

#Model performance on validation dataset
set.seed(442)
cv_predict.rose<-predict(cv_rose,as.matrix(valid.rose),s = "lambda.min", type = "class")
cv_predict.rose

#Confusion matrix
set.seed(478)
#actual target variable
target<-valid.rose$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.rose<-as.factor(cv_predict.rose)
#Confusion matrix
confusionMatrix(data=cv_predict.rose,reference=target)

#ROC_AUC score and curve
set.seed(843)
#convert to numeric
cv_predict.rose<-as.numeric(cv_predict.rose)
D=valid.rose
E=cv_predict.rose
D=cbind(D,E)
roc(data=D,response=target,predictor=E,auc=TRUE,plot=TRUE)
#Final submission
E=replace(E,E==1,0)
E=replace(E,E==2,1)
result<-data.frame(ID_code=df_test$ID_code,Prediction=E)
write.csv(result,'Final.CSV',row.names=F)
head(result)

#LightGBM:-LightGBM is a gradient boosting framework that uses tree based learning algorithms.
#We are going to use LightGBM model.

#Training and validation dataset

#Convert data frame to matrix
set.seed(5432)
#X_train<-as.matrix(train.data01[,-c(1,2)])
#y_train<-as.matrix(train.data01$target)
#X_valid<-as.matrix(valid.data01[,-c(1,2)])
#y_valid<-as.matrix(valid.data01$target)
#test_data<-as.matrix(df_test[,-c(1)])

#training data
#lgb.train <- lgb.Dataset(data=X_train, label=y_train)
#Validation data
```

```
#lgb.valid <- lgb.Dataset(data=X_valid,label=y_valid)

#Choosing best hyperparameters

#Selecting best hyperparameters
#set.seed(653)
#lgb.grid = list(objective = "binary",
                #metric = "auc",
               # boost='gbdt',
                #max_depth=-1,
                #boost_from_average='false',
                #min_sum_hessian_in_leaf = 12,
               # feature_fraction = 0.05,
               # bagging_fraction = 0.45,
               # bagging_freq = 5,
               # learning_rate=0.02,
               # tree_learner='serial',
               # num_leaves=20,
               # num_threads=5,
               # min_data_in_bin=150,
                #min_gain_to_split = 30,
               # min_data_in_leaf = 90,
                #verbosity=-1,
               # is_unbalance = TRUE)

#Training the lgbm model

#set.seed(7663)
#lgbm.model <- lgb.train(params = lgb.grid, data = lgb.train, nrounds =10000,eval_freq =1000,
                    valids=list(val1=lgb.train,val2=lgb.valid),early_stopping_rounds = 5000)
#lgbm model performance on test data
#set.seed(6532)
#lgbm_pred_prob <- predict(lgbm.model,test_data)
#print(lgbm_pred_prob)
#Convert to binary output (1 and 0) with threshold 0.5
#lgbm_pred<-ifelse(lgbm_pred_prob>0.5,1,0)
#print(lgbm_pred)

#Let us plot the important features
#set.seed(6521)
#feature importance plot
#tree_imp <- lgb.importance(lgbm.model, percentage = TRUE)
#lgb.plot.importance(tree_imp, top_n = 50, measure = "Frequency", left_margin = 10)

#We tried model with logistic regression,ROSE and lightgbm. But,lightgbm is performing well on
imbalanced data compared to other models based on scores of roc_auc_score.

#Final submission
#Final<-
#data.frame(ID_code=df_test$ID_code,lgb_predict_prob=lgbm_pred_prob,lgb_predict=lgbm_pred)
#write.csv(Final,'Final-R.CSV',row.names=F)
```