**Santander Customer Transaction Prediction**

**By**

**Mitta Jitendar Reddy**

**Introduction**

**Chapter 1:**

* Problemstatement
* Data

**Chapter 2: Methodology**

* Exploratory Data Analysis
* Missing Value Analysis
* Distributions of attributes
* Outlier Analysis

**Chapter 3: Modelling**

* Logistic Regression
* Decision Trees
* Random Forest
* LGBM

**Chapter 4: Model Evaluation**

* Model Selection

**Appendix**

**Santander Customer Transaction**

**Problem Statement**

At Santander, mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals.

Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

In this challenge, we need to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

**Data**

In this project, our task is to build classification models which will be used to predict which customers will make a specific transaction in the future.

We have train data and test data.

**Train**

In train data there are total 202 columns and 200000 rows

Names of the columns are:

ID\_code

target

var\_0

var\_1

var\_2

...

...

var\_195

var\_196

var\_197

var\_198

var\_199

**Test**

In test data we have 201 columns. Names of the columns are :

Id\_code

var\_0

var\_1

var\_2

...

...

var\_195

var\_196

var\_197

var\_198

var\_199

**Chapter 2**

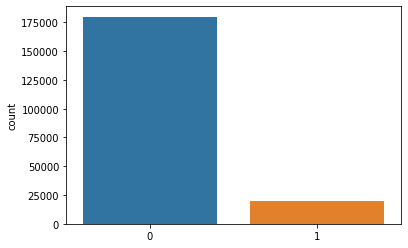
**Methodology**

**Exploratory Data Analysis (EDA)**

In statistics, exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

**Checking data is balanced or imbalanced**

Plotting the target class



From the above plot we can know that there is large difference in the number between class 0 and class 1.

Class 0 has around 90% of data and class 1 has only 10 % of data. So the data is imbalanced data.

**Missing value Analysis**

In missing value analysis we find if there are any missing cells present in the data or not. If there is any data then we need to fill that using various techniques like mean, median, KNNImputation etc.

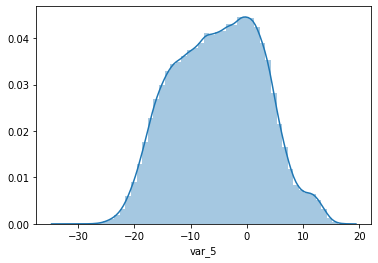
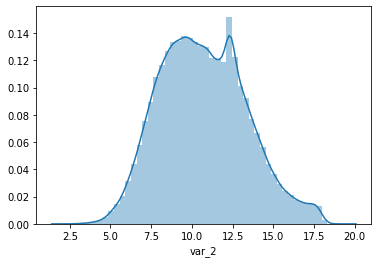
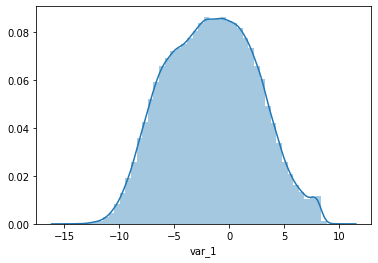
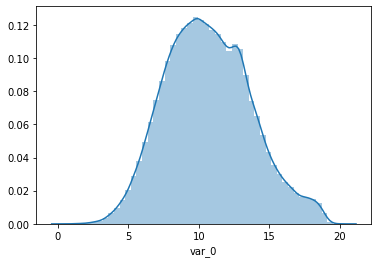
We find missing values in the data using

-> data.isna().sum()

On looking into the data there is no missing value present in the data.

**Distributions of variables**

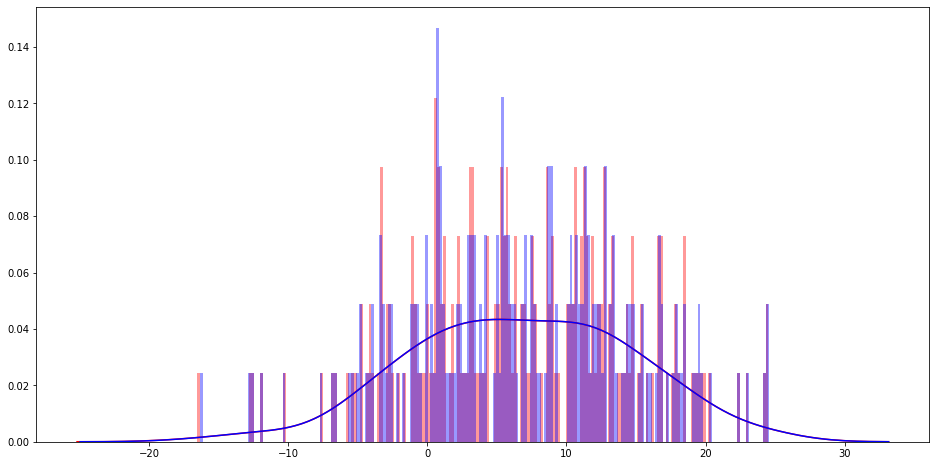
Let us look few of the variables distribution



All the variables are nearly skewed and there is no need to transform into guassian. If the data is more left or right skewed then the weights are tends towards the skewness side. This will effect greatly on accuracy.

**Distribution of mean values in both train and test dataset:-**

Below is the plot of means of all variables in train and test data except target variable

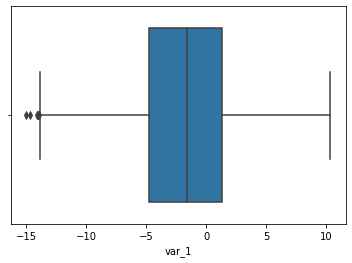
i

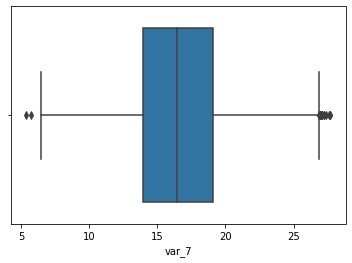
From the above plot we can know that there is no much new data in test data. This means in test data the data which is not in train is there in test data but it is not totally new data.

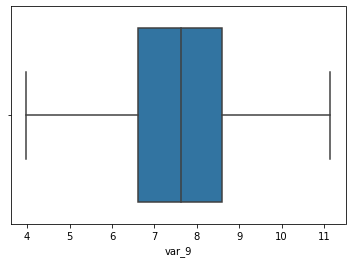
**Outlier analysis**

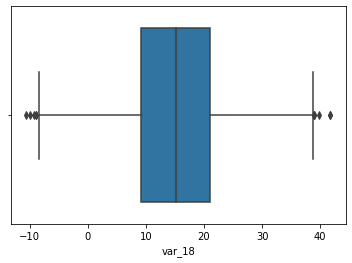
In our data the data is imbalanced data so we need to be careful while analyzing outliers.

Below are the box plots of some variables:









Above are the plots of some variables on analyzing those plots on performing box plot it is showing there are few outliers but on going into deep, the outliers it is showing are not far away from the whiskers. We don’t consider those are outliers. So in this data there are no outliers.

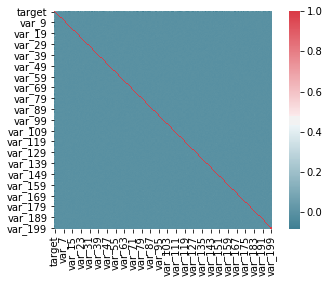
**Feature Selection**

Feature selection is very important for modelling the dataset. The every dataset have good and unwanted features. The unwanted features would effect on performance of model, so we have to delete those features. We have to select best features by using ANOVA, Chi-Square test and correlation matrix statistical techniques and so on. In this, we are selecting best features by using Correlation matrix.

**Correlation**

In statistics, correlation or dependence is any statistical relationship, whether causal or not, between two random variables or bivariate data.

The below is the correlation plot of the data:



On looking into the above graph it looks like there is no correlation among the variables. That means all variables are independent to each other.

And there is another point, no single variable is greatly predict the target variable.

**Feature engineering**

rf\_model=RandomForestClassifier(n\_estimators=10,random\_state=42)

rf\_model.fit(X\_test,y\_test)

from eli5.sklearn import PermutationImportance

perm\_imp=PermutationImportance(rf\_model,random\_state=39)

perm\_imp.fit(X\_test,y\_test)

eli5.show\_weights(perm\_imp,feature\_names=X\_test.columns.tolist(),top=200)

By running of above code we can know the Features which are important, that means which will greatly effect on finding the target class.

By running the code we came to know the most important features are:

Var\_81, Var\_146, Var\_109, Var\_12, Var\_110, Var\_173, Var\_174 …...etc

**Dealing of Imbalanced data**

We know that our data is imbalanced so we need to take care of that unless our model is biased towards the class which has more frequency.

To handle imbalanced data there are various techniques two of the most popular is up sampling and down sampling of data.

Down sampling means the class which have high count reduce to the count of class which have less frequency. Down sampling is not the best technique as we loss the data which is costly.

In the case of up sampling the class which have low frequency increase the count equal to class containing high.

We can do that in python using

class\_0=data[data['target']==0]

class\_1=data[data['target']==1]

class\_1=resample(class\_1,n\_samples=len(class\_0),replace=True)

data=pd.concat([class\_0,class\_1])

With this our new data is balanced

**Modeling**

Our target is to find the class of a record which means our data is belongs to classification type. We need to apply the classification models to train the data.

For training of the models first we need to split the data into two parts one part contains the data we need to train other is used for validation.

**Logistic Regression**

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist.

We train the model using below code - python:

LR=LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=20000,

multi\_class='auto', n\_jobs=None, penalty='l2',

random\_state=42, solver='lbfgs', tol=0.0001, verbose=0,

warm\_start=False)

LR.fit(X\_train,Y\_train)

Now find the accuracy of model on train data and validation data

Y\_pre=LR.predict(X\_train)

print(pd.crosstab(Y\_train,Y\_pre))

print(roc\_auc\_score(Y\_train,Y\_pre))

From this model we got train accuracy is 0.78

|  |  |  |
| --- | --- | --- |
| target | 0 | 1 |
| 0 | 112545 | 31430 |
| 1 | 31738 | 112130 |

Precision is 112130/(112130+31738)=0.78

Recall is 112130/(112130+31430)=0.79

Which is too low. We need to increase our accuracy.

**Decision Tree Classifier**

A decision tree classifier is a tree in which internal nodes are labeled by features. This model is trained using below code.

DTC=DecisionTreeClassifier(criterion='gini',

max\_depth=None, max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort='deprecated',

random\_state=None, splitter='best')

DTC.fit(X\_train,Y\_train)

After train the accuracy of model on train data is 1.0 which is overfit.

|  |  |  |
| --- | --- | --- |
| target | 0 | 1 |
| 0 | 143975 | 0 |
| 1 | 0 | 143975 |

But on test data it is low the accuracy is 0.94

|  |  |  |
| --- | --- | --- |
| target | 0 | 1 |
| 0 | 31650 | 4277 |
| 1 | 16 | 36018 |

Precision is 36018/(16+36018)=0.99

Recall is 36018/(36018+4277)= 0.89

**Random Forest Classifier**

Random forest classifier is an ensemble method. On training our data using random forest using below code in python:

rfc=RandomForestClassifier(criterion='gini', max\_depth=None, max\_features='auto',

max\_leaf\_nodes=None, max\_samples=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=100)

rfc.fit(X\_train,Y\_train)

This model accuracy on train data is 1.0

|  |  |  |
| --- | --- | --- |
| target | 0 | 1 |
| 0 | 143975 | 0 |
| 1 | 0 | 143975 |

Accuracy on test data is 0.999

|  |  |  |
| --- | --- | --- |
| target | 0 | 1 |
| 0 | 35927 | 0 |
| 1 | 16 | 36018 |

Precision is 36018/(36018+16) =0.99

Recall is 36018/36018 = 1.0

Random forest is performing both on train and validation data.

**LGBM**

Lgbm is also ensemble method, this is the modified method of gradient boosting method.

--------------python

lgb\_train=lgb.Dataset(X\_train,label=Y\_train)

lgb\_valid=lgb.Dataset(X\_test,label=Y\_test)

params={'boosting\_type': 'gbdt',

'max\_depth' : -1,

'objective': 'binary',

'boost\_from\_average':False,

'nthread': 20,

'metric':'auc',

'num\_leaves': 50,

'learning\_rate': 0.01,

'max\_bin': 100,

'subsample\_for\_bin': 100,

'subsample': 1,

'subsample\_freq': 1,

'colsample\_bytree': 0.8,

'bagging\_fraction':0.5,

'bagging\_freq':5,

'feature\_fraction':0.08,

'min\_split\_gain': 0.45,

'min\_child\_weight': 1,

'min\_child\_samples': 5,

'is\_unbalance':True,

}

num\_rounds=20000

lgbm= lgb.train(params,lgb\_train,num\_rounds,valid\_sets=[lgb\_train,lgb\_valid],verbose\_eval=1000,early\_stopping\_rounds = 5000)

lgbm

On using above code we train our model . Here we set the num\_rounds to 20000 which will get increase the accuracy for every round it outputs the the round which has great accuracy.

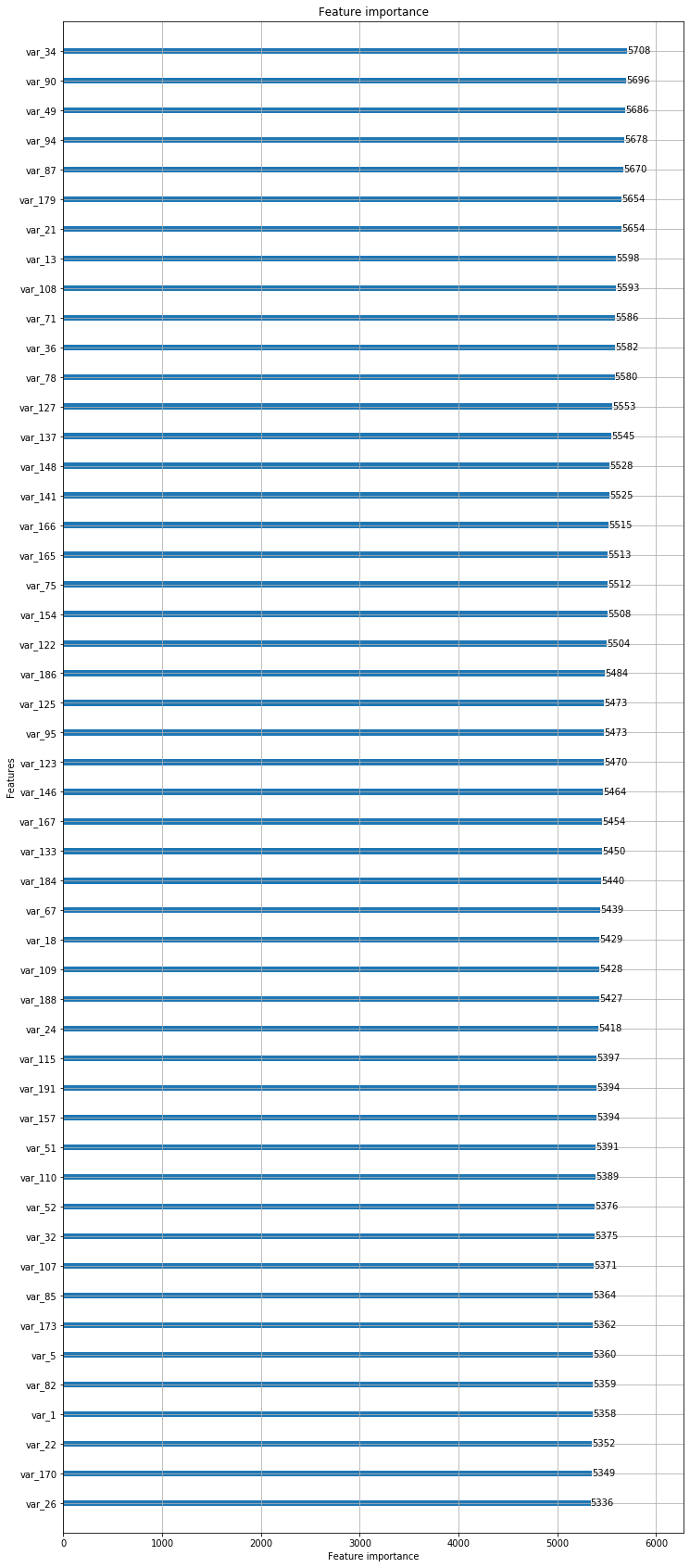
This model has accuracy on train data is 0.9999

This model has accuracy on test data is 0.9974

This is better model compare to the Random forest

From this model we can know the importance of each variable.

The below is the plot of importance of the variables.



**Model Evaluation**

From the all models we train below table is the accuracy on train and test data.

|  |  |  |
| --- | --- | --- |
| Model | Train accuracy | Test accuracy |
| Logistic Regression | 0.78 | 0.64 |
| Decision Tree | 1.0 | 0.94 |
| Random Forest | 1.0 | 0.98 |
| LGBM | 0.9999 | 0.9997 |

|  |  |  |
| --- | --- | --- |
| Model | Precision | Recall |
| Logistic Regression | 0.78 | 0.79 |
| Decision Tree | 0.99 | 0.89 |
| Random Forest | 0.991 | 0.997 |
| LGBM | 0.999 | 1.0 |

From the data of all models Decision tree and Random forest is overfit the data so there test accuracy is less.

Among all the models LGBM is good at both train and test accuracy.

So we finalize the LGBM and train all the data using LGBM.

#probability predictions

lgbm\_predict\_prob=lgbm.predict(final\_test.iloc[:,1:],random\_state=39,num\_iteration=lgbm.best\_iteration)

#Convert to binary output 1 or 0

lgbm\_predict=np.where(lgbm\_predict\_prob>=0.5,1,0)

print(lgbm\_predict\_prob)

print(lgbm\_predict)

After we train the model apply on final test data and save into csv file.

**Appendix**

**Python code**

# -\*- coding: utf-8 -\*-

"""# Load Required Libraries"""

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import roc\_auc\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import GridSearchCV

from sklearn.utils import resample

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model\_selection import StratifiedKFold

import lightgbm as lgb

"""#Load Data"""

data=pd.read\_csv("/content/drive/My Drive/train.csv")

final\_test=pd.read\_csv("/content/drive/My Drive/test.csv")

data.head()

"""# EXPLORATORY DATA ANALYSIS"""

data['target'].value\_counts()

sns.countplot(data.target.values)

"""In the target there are two classes (0,1). The data belongs to class 0 count is too large than class 1. From this we consider data is imbalanced data"""

data.isna().sum()

data.isna().sum().sum()

"""From the analysis in the data there are no null values"""

for i in data.columns[2:]:

sns.distplot(data[i])

plt.show()

"""On validating distribution of each variable in data the data has not much skewed,nearly symmetric about a mean"""

data.dtypes

data.describe()

data.corr()

corr=data.corr()

f,ax=plt.subplots()

sns.heatmap(corr,mask=np.zeros\_like(corr,dtype=np.bool),cmap=sns.diverging\_palette(220,10,as\_cmap=True),square=True,ax=ax)

"""# Outlier Analysis

It looks like there is no correlation between variables in the data

"""

for i in range(2,len(data.columns)):

sns.boxplot(x=data[data.columns[i]])

plt.show()

"""Using boxplot we can know the outliers. In the above pictures box plot is showing few points are outliers but those points are not wide away from the whiskers.Those are within the range of 2-5 units. So we dont consider as outliers."""

train\_attr=data.columns[2:]

test\_attr=final\_test.columns[1:]

plt.figure(figsize=(16,8))

#Distribution plot for mean values per column in train attributes

sns.distplot(data[train\_attr].mean(axis=0),color='red',kde=True,bins=200,label='train')

#Distribution plot for mean values per column in test attributes

sns.distplot(final\_test[test\_attr].mean(axis=0),color='blue',kde=True,bins=200,label='test')

plt.show()

"""#Dealing of imbalanced data"""

class\_0=data[data['target']==0]

class\_1=data[data['target']==1]

class\_1=resample(class\_1,n\_samples=len(class\_0),replace=True)

data=pd.concat([class\_0,class\_1])

"""Now the class 0 and class 1 have same count"""

data.shape

data['target'].value\_counts()

"""# Model Training"""

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(data.iloc[:,2:],data.iloc[:,1],test\_size=0.2)

print(X\_train.shape)

print(Y\_train.shape)

print(X\_test.shape)

print(Y\_test.shape)

"""# Logistic Regression"""

LR=LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=20000,

multi\_class='auto', n\_jobs=None, penalty='l2',

random\_state=42, solver='lbfgs', tol=0.0001, verbose=0,

warm\_start=False)

LR.fit(X\_train,Y\_train)

Y\_pre=LR.predict(X\_train)

print(pd.crosstab(Y\_train,Y\_pre))

print(roc\_auc\_score(Y\_train,Y\_pre))

"""# Decision Tree Classifier"""

DTC=DecisionTreeClassifier(criterion='gini',

max\_depth=None, max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort='deprecated',

random\_state=None, splitter='best')

DTC.fit(X\_train,Y\_train)

Y\_pre=DTC.predict(X\_train)

print(pd.crosstab(Y\_train,Y\_pre))

print(roc\_auc\_score(Y\_train,Y\_pre))

Y\_pre=DTC.predict(X\_test)

print(pd.crosstab(Y\_test,Y\_pre))

print(roc\_auc\_score(Y\_test,Y\_pre))

"""In decision tree the test accuracy is more difference than train accuracy. It seems like it was overfitting.

# Random Forest

"""

rfc=RandomForestClassifier(criterion='gini', max\_depth=None, max\_features='auto',

max\_leaf\_nodes=None, max\_samples=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=100)

rfc.fit(X\_train,Y\_train)

Y\_pre=rfc.predict(X\_train)

print(pd.crosstab(Y\_train,Y\_pre))

print(roc\_auc\_score(Y\_train,Y\_pre))

Y\_pre=rfc.predict(X\_test)

print(pd.crosstab(Y\_test,Y\_pre))

print(roc\_auc\_score(Y\_test,Y\_pre))

"""Random Forest has great test accuracy than Decision tree

# Gradient Boosting

"""

gbm=GradientBoostingClassifier(loss='deviance', learning\_rate=0.1, n\_estimators=100, subsample=1.0, criterion='friedman\_mse', min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_depth=7, min\_impurity\_decrease=0.0, min\_impurity\_split=None, init=None, random\_state=None, max\_features=None, verbose=2, max\_leaf\_nodes=None, warm\_start=False, presort='deprecated', validation\_fraction=0.1, n\_iter\_no\_change=None, tol=0.0001, ccp\_alpha=0.0)

gbm.fit(X\_train,Y\_train)

Y\_pre=gbm.predict(X\_train)

print(pd.crosstab(Y\_train,Y\_pre))

print(roc\_auc\_score(Y\_train,Y\_pre))

"""# LGBM"""

#Training data

lgb\_train=lgb.Dataset(X\_train,label=Y\_train)

#Validation data

lgb\_valid=lgb.Dataset(X\_test,label=Y\_test)

params={'boosting\_type': 'gbdt',

'max\_depth' : -1,

'objective': 'binary',

'boost\_from\_average':False,

'nthread': 20,

'metric':'auc',

'num\_leaves': 50,

'learning\_rate': 0.01,

'max\_bin': 100,

'subsample\_for\_bin': 100,

'subsample': 1,

'subsample\_freq': 1,

'colsample\_bytree': 0.8,

'bagging\_fraction':0.5,

'bagging\_freq':5,

'feature\_fraction':0.08,

'min\_split\_gain': 0.45,

'min\_child\_weight': 1,

'min\_child\_samples': 5,

'is\_unbalance':True,

}

num\_rounds=20000

lgbm= lgb.train(params,lgb\_train,num\_rounds,valid\_sets=[lgb\_train,lgb\_valid],verbose\_eval=1000,early\_stopping\_rounds = 5000)

lgbm

#probability predictions

lgbm\_predict\_prob=lgbm.predict(X\_test,random\_state=42,num\_iteration=lgbm.best\_iteration)

#Convert to binary output 1 or 0

lgbm\_predict=np.where(lgbm\_predict\_prob>=0.5,1,0)

print(lgbm\_predict\_prob)

print(lgbm\_predict)

lgb.plot\_importance(lgbm,max\_num\_features=50,importance\_type="split",figsize=(12,30))

"""# Model Evaluation

From Logistic regression, Decision Tree, Random Forest the best model is LGBM.

"""

data.shape

data.head()

final\_test.head()

#probability predictions

lgbm\_predict\_prob=lgbm.predict(final\_test.iloc[:,1:],random\_state=39,num\_iteration=lgbm.best\_iteration)

#Convert to binary output 1 or 0

lgbm\_predict=np.where(lgbm\_predict\_prob>=0.5,1,0)

print(lgbm\_predict\_prob)

print(lgbm\_predict)

fin=pd.DataFrame({'ID\_code':final\_test['ID\_code'].values})

fin['target']=lgbm\_predict

fin.to\_csv('fsubmission.csv',index=False)

fin.head()

**R Code**

rm(list=ls(all=T))

#Loading Libraries:-

library(tidyverse)

library(moments)

library(DataExplorer)

library(caret)

library(Matrix)

library(pdp)

library(mlbench)

library(caTools)

library(randomForest)

library(glmnet)

library(mlr)

library(vita)

library(rBayesianOptimization)

library(lightgbm)

library(pROC)

library(DMwR)

library(ROSE)

library(yardstick)

#Setting Directory:-

setwd("C:/Users/mitta/Downloads")

#Importing the training Data:-

df\_train=read.csv("train.csv")

head(df\_train)

class(df\_train)

#Dimension of the train data:-

dim(df\_train)

#Summary of the train dataset:-

str(df\_train)

#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 taregt class in train data:-

table(df\_train$target)/length(df\_train$target)\*100

#Bar plot for count of target classes in train data:-

ggplot(df\_train,aes(target))+theme\_bw()+geom\_bar(stat='count',fill='red')

#Observation:- We are having a unbalanced data, where 90% of the data is no. of customers who will not make a transaction & 10 % of the data are those who will make a transaction.

#Distribution of train attributes from 3 to 202:-

for (var in names(df\_train)[c(3:202)]){

target<-df\_train$target

plot<-ggplot(df\_train, aes(df\_train[[var]],fill=target)) +

geom\_density(kernel='gaussian') + ggtitle(var)+theme\_classic()

print(plot)

}

#Importing the test data:-

df\_test=read.csv("test.csv")

head(df\_test)

#Dimension of test dataset:-

dim(df\_test)

#Distribution of test attributes from 2 to 201:-

plot\_density(df\_test[,c(2:201)],ggtheme = theme\_classic(),geom\_density\_args = list(color='red'))

#Mean value per rows and columns in train & test dataset:-

#Applying the function to find mean values per row in train and test data.

train\_mean<-apply(df\_train[,-c(1,2)],MARGIN=1,FUN=mean)

test\_mean<-apply(df\_test[,-c(1)],MARGIN=1,FUN=mean)

ggplot()+

#Distribution of mean values per row in train data

geom\_density(data=df\_train[,-c(1,2)],aes(x=train\_mean),kernel='gaussian',show.legend=TRUE,color='blue')+theme\_classic()+

#Distribution of mean values per row in test data

geom\_density(data=df\_test[,-c(1)],aes(x=test\_mean),kernel='gaussian',show.legend=TRUE,color='green')+

labs(x='mean values per row',title="Distribution of mean values per row in train and test dataset")

#Applying the function to find mean values per column in train and test data.

train\_mean<-apply(df\_train[,-c(1,2)],MARGIN=2,FUN=mean)

test\_mean<-apply(df\_test[,-c(1)],MARGIN=2,FUN=mean)

ggplot()+

#Distribution of mean values per column in train data

geom\_density(aes(x=train\_mean),kernel='gaussian',show.legend=TRUE,color='blue')+theme\_classic()+

#Distribution of mean values per column in test data

geom\_density(aes(x=test\_mean),kernel='gaussian',show.legend=TRUE,color='green')+

labs(x='mean values per column',title="Distribution of mean values per column in train and test dataset")

#Applying the function to find standard deviation values per row in train and test data.

train\_sd<-apply(df\_train[,-c(1,2)],MARGIN=1,FUN=sd)

test\_sd<-apply(df\_test[,-c(1)],MARGIN=1,FUN=sd)

ggplot()+

#Distribution of sd values per row in train data

geom\_density(data=df\_train[,-c(1,2)],aes(x=train\_sd),kernel='gaussian',show.legend=TRUE,color='red')+theme\_classic()+

#Distribution of sd values per row in test data

geom\_density(data=df\_test[,-c(1)],aes(x=test\_sd),kernel='gaussian',show.legend=TRUE,color='blue')+

labs(x='sd values per row',title="Distribution of sd values per row in train and test dataset")

#Applying the function to find sd values per column in train and test data.

train\_sd<-apply(df\_train[,-c(1,2)],MARGIN=2,FUN=sd)

test\_sd<-apply(df\_test[,-c(1)],MARGIN=2,FUN=sd)

ggplot()+

#Distribution of sd values per column in train data

geom\_density(aes(x=train\_sd),kernel='gaussian',show.legend=TRUE,color='red')+theme\_classic()+

#Distribution of sd values per column in test data

geom\_density(aes(x=test\_sd),kernel='gaussian',show.legend=TRUE,color='blue')+

labs(x='sd values per column',title="Distribution of std values per column in train and test dataset")

#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

#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

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

#Feature Enginnering:- Performing some feature engineering on datasets:-

#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

VarImp<-importance(rf,type=2)

VarImp

#Observation:-We can observed that the top important features are var\_12, var\_26, var\_22,v var\_174, var\_198 and so on based on Mean decrease gini.

#Partial dependence plots:-PDP gives a graphical depiction of marginal effect of a variable on the class probability or classification. It shows how a feature effects predictions.

#Calculation of partial dependence plots on random forest:-

#we are observing impact of main features which are discovered in previous section by using PDP Plot.

#We will plot "var\_13"

par.var\_13 <- partial(rf, pred.var = c("var\_13"), chull = TRUE)

plot.var\_13 <- autoplot(par.var\_13, contour = TRUE)

plot.var\_13

#We will plot "var\_6"

par.var\_6 <- partial(rf, pred.var = c("var\_6"), chull = TRUE)

plot.var\_6 <- autoplot(par.var\_6, contour = TRUE)

plot.var\_6

#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.data<-df\_train[train.index,]

#validation data

valid.data<-df\_train[-train.index,]

#dimension of train data

dim(train.data)

#dimension of validation data

dim(valid.data)

#target classes in train data

table(train.data$target)

#target classes in validation data

table(valid.data$target)

#Training and validation dataset

#Training dataset

X\_t<-as.matrix(train.data[,-c(1,2)])

y\_t<-as.matrix(train.data$target)

#validation dataset

X\_v<-as.matrix(valid.data[,-c(1,2)])

y\_v<-as.matrix(valid.data$target)

#test dataset

test<-as.matrix(df\_test[,-c(1)])

#Logistic regression model

set.seed(667) # to reproduce results

lr\_model <-glmnet(X\_t,y\_t, family = "binomial")

summary(lr\_model)

#Cross validation prediction

set.seed(8909)

cv\_lr <- cv.glmnet(X\_t,y\_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,X\_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.data$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)

#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.data[,-c(1)],seed=32)$data

#target classes in balanced train data

table(train.rose$target)

valid.rose <- ROSE(target~., data =valid.data[,-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)

roc(data=valid.rose[,-c(1,2)],response=target,predictor=cv\_predict.rose,auc=TRUE,plot=TRUE)

#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.data[,-c(1,2)])

y\_train<-as.matrix(train.data$target)

X\_valid<-as.matrix(valid.data[,-c(1,2)])

y\_valid<-as.matrix(valid.data$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

sub\_df<-data.frame(ID\_code=df\_test$ID\_code,lgb\_predict\_prob=lgbm\_pred\_prob,lgb\_predict=lgbm\_pred)

write.csv(sub\_df,'submission-R.CSV',row.names=F)

head(sub\_df)