**Report**

**Santander Customer Transaction**

**Prediction**

**MANIDEEP**

**Student of Edwisor**

**Contents**

* Problem Statement
* Identification of problem
* Data pre processing
* Model development
* Evaluation metrics
* Results
* R code
* Python Code

**Problem Statement :**

In this challenge, we need to identify which customers will make a specific transaction in

the future, irrespective of the amount of money transacted.

**Identification of Problem :**

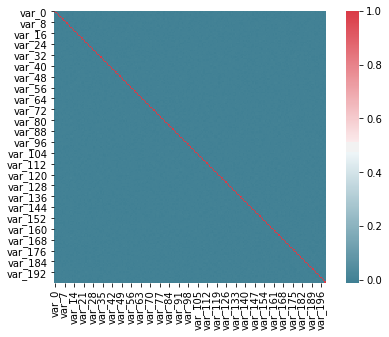
According to the problem statement, we need to identify the specific customers who will make a transaction. Therefore, the problem type is classification(Yes/No).

**Data pre processing** :

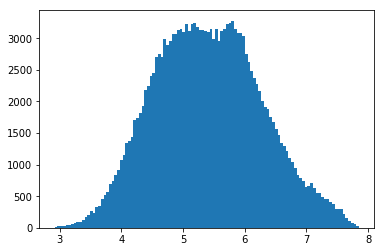
Two excel sheets are given namely train and test data. The below mentioned techniques are applied using R and Python on train data and test data(new data) :

1. Missing value analysis
2. Outlier Analysis
3. Scaling
4. Feature selection
5. Standardization

During feature selection correlation plot is examined carefully to detect the relation between independent variables.



All the variables are independent to each other. To detect whether the data is skewed or normally distributed a histogram is constructed. Since the data is normally distributed according to the below graph, standardization is used.



**Model development :**

Initially the process is started by developing logistic regression. In the document it is mentioned to implement 3 algorithms. So, I chose decision tree algorithm along with random forest and naïve bayes. Implemented all the algorithms in R and python.

**Evaluation Metrics** :

Metrics like AUC, Precison and recall are calculated using confusion matrix to detect the efficiency of the algorithms.

* AUC is generally used for the classification problems.
* Precision depicts the proportion of results which are relevant.
* Precision = tp/(tp + fp)
* Recall depicts the proportion of total relevant results correctly classified by the algorithm.
* Recall = tp/(tp + fn)

**Results** :

Results of evaluation metrics are used for detecting the algorithm which is more efficient.

**Python** :

**Logistic regression**:

Accuracy: 0.90825

Precision: 0.6167664670658682

Recall: 0.2536945812807882

AUC: 0.6179435622041114

**Decision tree**:

Accuracy: 0.82525

Precision: 0.16009280742459397

Recall: 0.16995073891625614

AUC: 0.5346136554904042

**Random Forest**:

Accuracy: 0.89875

Precision: 1.0

Recall: 0.0024630541871921183

AUC: 0.5012315270935961

**Naïve Bayes**:

Accuracy: 0.918

Precision: 0.6839622641509434

Recall: 0.35714285714285715

AUC: 0.6692503378646952

From these values it is found that Naïve bayes has higher AUC, accuracy and precision compared to other methods. So, I chose this method and predicted the values of test data.

**R** :

**Logistic regression**:

AUC: 1

Precision: 1

Recall: 1

**Decision tree**:

AUC: 0.5352

Precision: 0.99

Recall: 0.987

**Random Forest**:

AUC: 0.5

Precision: 1

Recall: 1

**Naïve Bayes**:

AUC: 0.642

Precision: 0.97

Recall: 0.984

From these values it is found that Logistic regression has higher AUC, accuracy and precision compared to other methods. So, I chose this method and predicted the values of test data.

**Code**:

**R**:

#Load the path

setwd("C:/manideep/edwisor/Project-1")

getwd()

#Load the data

test\_data = read.csv("test.csv", header = T)

train\_data = read.csv("train.csv", header = T)

#Required packages

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information",

"MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees')

lapply(x, require, character.only = TRUE)

library(glmnet)

#---------------------------------------------DATA PRE PROCESSING---------------------------------------------------

#sampling

test\_sample = test\_data[sample(nrow(test\_data), 20000, replace = F), ]

train\_sample = train\_data[sample(nrow(train\_data), 20000, replace = F), ]

# Missing value Analysis for test and train data

missing\_val = data.frame(apply(train\_sample,2,function(x){sum(is.na(x))}))

missing\_val$Columns = row.names(missing\_val)

row.names(missing\_val) = NULL

names(missing\_val)[1] = "percent"

missing\_val$percent = (missing\_val$percent/nrow(train\_sample)) \* 100

missing\_val = missing\_val[order(-missing\_val$percent),]

row.names(missing\_val) = NULL

missing\_val = missing\_val[,c(2,1)]

missing\_val1 = data.frame(apply(test\_sample,2,function(x){sum(is.na(x))}))

missing\_val1$Columns = row.names(missing\_val1)

row.names(missing\_val1) = NULL

names(missing\_val1)[1] = "percent"

missing\_val1$percent = (missing\_val1$percent/nrow(test\_sample)) \* 100

missing\_val1 = missing\_val1[order(-missing\_val1$percent),]

row.names(missing\_val1) = NULL

missing\_val1 = missing\_val1[,c(2,1)]

#Outlier Analysis for both train and test data

cnames = colnames(train\_sample[,3:202])

train\_sample1 = train\_sample

for(i in cnames){

print(i)

val = train\_sample[,i][train\_sample[,i] %in% boxplot.stats(train\_sample[,i])$out]

print(length(val))

train\_sample = train\_sample[which(!train\_sample[,i] %in% val),]

}

#Correlation matrix

tr = cor(train\_sample[,unlist(lapply(train\_sample, is.numeric))])

#Standardization

for(i in cnames){

print(i)

train\_sample[,i] = (train\_sample[,i] - mean(train\_sample[,i]))/

sd(train\_sample[,i])

}

#---------------------------------------------------- Algorithms----------------------------------------------------

#Divide data into train and test.

division = function(){

set.seed(123)

train.num = createDataPartition(train\_sample[,"target"], p = .80, list = FALSE)

train = train\_sample[ train.num,]

test = train\_sample[-train.num,]

}

library(dplyr)

#scaling

X\_train <- scale(train[,-(1:2)]) %>% data.frame

X\_test <- scale(test[,-1]) %>% data.frame

target <- train$target

library(speedglm)

#Logistic regression

logit\_model = speedglm(target ~ ., data = X\_train, family = binomial())

summary(logit\_model)

logit\_Predictions = predict(logit\_model, newdata = X\_test , type = "response")

logit\_Predictions = ifelse(logit\_Predictions > 0.5, 1, 0)

X\_test$target = logit\_Predictions

#Evaluation

ConfMatrix\_RF = table(X\_test$target, logit\_Predictions)

ConfMatrix\_RF

#0 3359 0

#1 0 150

#Metrics

library(pROC)

roc\_obj\_lr <- roc(X\_test$target, logit\_Predictions) #1

auc(roc\_obj\_lr)

Precision\_lr = 150/(150 + 0) #1 tp/(tp + fp)

recall\_lr = 150/(150 + 0) #1 tp/(tp + fn)

# Before running next algorithm run the lines 1 to 80.

# Decision Tree C50

train$target<-as.factor(train$target)

str(train$target)

scale\_fun = function(){

target <- train$target

Y\_train <- scale(train[,-(1:2)]) %>% data.frame

Y\_test <- scale(test[,-(1:2)]) %>% data.frame

Y\_test$target = test[,2]

}

C50\_model = C5.0(target ~., Y\_train, trials = 10, rules = TRUE)

#Summary of DT model

summary(C50\_model)

#write rules into disk

write(capture.output(summary(C50\_model)), "c50Rules.txt")

#Lets predict for test cases

C50\_Predictions = predict(C50\_model, Y\_test[,-201], type = "class")

write.csv(C50\_Predictions, "C50\_target.csv", row.names = T)

##Evaluate the performance of classification model

ConfMatrix\_C50 = table(Y\_test$target, C50\_Predictions)

confusionMatrix(ConfMatrix\_C50)

#C50\_Predictions

#0 1

#0 3145 41

#1 297 27

C50\_Predictions = as.numeric(as.character(C50\_Predictions))

roc\_obj\_c50 <- roc(test$target, C50\_Predictions)

auc(roc\_obj\_c50) #0.5352

Precision\_c50 = 3145/(3145 + 27) #0.99

recall\_c50 = 3145/(3145+41) #0.987

#Random Forest

division()

scale\_fun()

RF\_model = randomForest(target ~ ., Y\_train, importance = TRUE, ntree = 500)

RF\_Predictions = predict(RF\_model, Y\_test[,-201])

write.csv(RF\_Predictions, "RF\_target.csv", row.names = T)

##Evaluate the performance of classification model

ConfMatrix\_RF = table(Y\_test$target, RF\_Predictions)

confusionMatrix(ConfMatrix\_RF)

#RF\_Predictions

#0 1

#0 3186 0

#1 324 0

RF\_Predictions = as.numeric(as.character(RF\_Predictions))

roc\_obj\_RF <- roc(Y\_test$target, RF\_Predictions)

auc(roc\_obj\_RF) #0.5

Precision\_RF = 3186/(3186 + 0) #1

recall\_RF = 3186/(3186+0) #1

library(e1071)

#Develop model

division()

scale\_fun()

NB\_model = naiveBayes(target ~ ., data = Y\_train)

#predict on test cases #raw

NB\_Predictions = predict(NB\_model, Y\_test[,1:200], type = 'class')

write.csv(NB\_Predictions, "NB\_target.csv", row.names = T)

#Look at confusion matrix

Conf\_matrix = table(observed = Y\_test[,201], predicted = NB\_Predictions)

confusionMatrix(Conf\_matrix)

#predicted

#observed 0 1

#0 3137 49

#1 227 97

NB\_Predictions = as.numeric(as.character(NB\_Predictions))

roc\_obj\_nb <- roc(Y\_test$target, NB\_Predictions)

auc(roc\_obj\_nb) #0.642

Precision\_nb = 3137/(3137 + 97) #0.970

recall\_nb = 3137/(3137+49) #0.984

#---------------------------------------------Prediction on new data---------------------------------

#New data

test\_data$ID\_code = NULL

pred = predict(logit\_model, newdata = test\_data,type = 'response')

pred = ifelse(pred > 0.5, 1, 0)

test\_data$target = pred

write.csv(pred, "final\_output\_r.csv", row.names = T)

**Python**:

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from random import randrange, uniform

os.chdir("C:\manideep\edwisor\Project-1")

os.getcwd()

train\_data = pd.read\_csv("train.csv")

test\_data = pd.read\_csv("test.csv")

#Create dataframe with missing percentage

missing\_val = pd.DataFrame(train\_data.isnull().sum())

missing\_val1 = pd.DataFrame(test\_data.isnull().sum())

#Reset index

missing\_val = missing\_val.reset\_index()

missing\_val1 = missing\_val1.reset\_index()

#Rename variable

missing\_val = missing\_val.rename(columns = {'index': 'Variables', 0: 'percent'})

missing\_val1 = missing\_val1.rename(columns = {'index': 'Variables', 0: 'percent'})

#Calculate percentage

missing\_val['percent'] = (missing\_val['percent']/len(train\_data))\*100

missing\_val1['percent'] = (missing\_val1['percent']/len(test\_data))\*100

cnames = train\_data.iloc[:,2:203]

cnames1 = test\_data.iloc[:,2:202]

for i in cnames:

print(i)

q75, q25 = np.percentile(train\_data.loc[:,i], [75 ,25])

iqr = q75 - q25

min = q25 - (iqr\*1.5)

max = q75 + (iqr\*1.5)

print(min)

print(max)

train\_data = train\_data.drop(train\_data[train\_data.loc[:,i] < min].index)

train\_data = train\_data.drop(train\_data[train\_data.loc[:,i] > max].index)

for i in cnames1:

print(i)

q75, q25 = np.percentile(test\_data.loc[:,i], [75 ,25])

iqr = q75 - q25

min = q25 - (iqr\*1.5)

max = q75 + (iqr\*1.5)

print(min)

print(max)

test\_data = test\_data.drop(test\_data[test\_data.loc[:,i] < min].index)

test\_data = test\_data.drop(test\_data[test\_data.loc[:,i] > max].index)

train\_corr = train\_data.iloc[:,2:202]

#Set the width and hieght of the plot

f, ax = plt.subplots(figsize=(7, 5))

#Generate correlation matrix

corr = train\_corr.corr()

#Plot using seaborn library

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

square=True, ax=ax)

%matplotlib inline

plt.hist(train\_data['var\_6'], bins='auto')

for i in cnames:

print(i)

train\_data[i] = (train\_data[i] - train\_data[i].mean())/train\_data[i].std()

for i in cnames1:

print(i)

test\_data[i] = (test\_data[i] - test\_data[i].mean())/test\_data[i].std()

train\_sample = train\_data.sample(20000)

from sklearn import tree

from sklearn.metrics import accuracy\_score

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

from sklearn.linear\_model import LogisticRegression

from sklearn import metrics

X = train\_sample.values[:, 2:202]

Y = train\_sample.values[:,1]

Y=Y.astype('int')

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, Y, test\_size = 0.2)

log\_reg = LogisticRegression()

# fit the model with data

log\_reg.fit(X\_train,y\_train)

#

y\_pred=log\_reg.predict(X\_test)

cnf\_matrix = metrics.confusion\_matrix(y\_test, y\_pred)

cnf\_matrix

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))

print("Precision:",metrics.precision\_score(y\_test, y\_pred))

print("Recall:",metrics.recall\_score(y\_test, y\_pred))

print("AUC:", metrics.roc\_auc\_score(y\_test, y\_pred))

C50\_model = tree.DecisionTreeClassifier(criterion='entropy').fit(X\_train, y\_train)

#predict new test cases

C50\_Pred = C50\_model.predict(X\_test)

cnf\_matrix\_dt = metrics.confusion\_matrix(y\_test, C50\_Pred)

cnf\_matrix\_dt

print("Accuracy:",metrics.accuracy\_score(y\_test, C50\_Pred))

print("Precision:",metrics.precision\_score(y\_test, C50\_Pred))

print("Recall:",metrics.recall\_score(y\_test, C50\_Pred))

print("AUC:", metrics.roc\_auc\_score(y\_test, C50\_Pred))

from sklearn.ensemble import RandomForestClassifier

RF\_model = RandomForestClassifier(n\_estimators = 20).fit(X\_train, y\_train)

RF\_Pred = RF\_model.predict(X\_test)

cnf\_matrix\_rf = metrics.confusion\_matrix(y\_test, RF\_Pred)

cnf\_matrix\_rf

print("Accuracy:",metrics.accuracy\_score(y\_test, RF\_Pred))

print("Precision:",metrics.precision\_score(y\_test, RF\_Pred))

print("Recall:",metrics.recall\_score(y\_test, RF\_Pred))

print("AUC:", metrics.roc\_auc\_score(y\_test, RF\_Pred))

from sklearn.naive\_bayes import GaussianNB

#Naive Bayes implementation

NB\_model = GaussianNB().fit(X\_train, y\_train)

NB\_Pred = NB\_model.predict(X\_test)

cnf\_matrix\_nb = metrics.confusion\_matrix(y\_test, NB\_Pred)

cnf\_matrix\_nb

print("Accuracy:",metrics.accuracy\_score(y\_test, NB\_Pred))

print("Precision:",metrics.precision\_score(y\_test, NB\_Pred))

print("Recall:",metrics.recall\_score(y\_test, NB\_Pred))

print("AUC:", metrics.roc\_auc\_score(y\_test, NB\_Pred))

x\_test = test\_data.drop(['ID\_code'],axis=1)

pred = NB\_model.predict(x\_test)

x\_test['target'] = pred

x\_test.to\_csv("final\_out.csv", index = False)