PROJECT – SANTANDER CUSTOMER TRANSACTION PREDICTION

By Anchit Chaturvedi

INDEX

INTRODUCTION

Background: 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?

Problem Statement: There is a dataset available from Santander company, about the amount of money transacted by the various customers. The task in hand is to predict whether a customer will make a certain monetary transaction or not, irrespective of the money transacted.

Data: The dataset available consists of data about monetary transactions done by the customers. It also contains a target column which tells whether a certain customer made a transaction or not. The customer is determined by their 'customer ID'. The target column has the value '0' or '1'. Also, there are 200 other numerical variables containing information about the money transacted.

DATA PRE-PROCESSING

In the pre-processing stage, there are various methodologies that are used. The data is checked for outliers, and to check whether there are missing values or not. The data types of all the variables is also corrected and converted to an appropriate type for use in the algorithms.

For the use in this methodology, the column 'customer ID' is not required. So, this column is not of any use for the process. Thus, first step is to drop this column from the data.

After this, we need to check the data types of the various columns. It should be of numeric type for use in the process. All these columns need to be converted to numeric types. The target column is a categorical variable, containing only the values '0' or '1'. Thus, this column needs to be converted to a categorical type, instead of a numeric type.

Missing Value Analysis

Missing values are null values in the data. They do not carry any useful information for the analysis. These values can either be dropped from the data altogether or they can also be imputed with appropriate values, using the following methods:

- Mean value imputation: The missing values are imputed by using the mean of all the other non-null values.
- Median value imputation: Here, the median of all the other non-null values is calculated. And then this median value is imputed to all the missing values.
- kNN imputation: This method is similar to the k-Nearest neighbors algorithm. Here, we try to find 'k' data points that are the closest to the missing values. Then, according to the nearest values, the missing value is imputed.

Outlier Analysis

Outliers are the values in the data, that are extremely larger or extremely smaller than the normal values that are there in the data. These values cause the mean of the data to be very large or very small. To find the outliers in the data, we need to find the inner and the outer boundaries of the data. For this, the various quartiles (the 25^{th} and the 75^{th}) need to be calculated. Once the quartiles are calculated, we can calculate the inner and the outer boundaries in the data. The values lying outside these boundaries are the outliers in the data. To treat these values, we can either drop these values or make them to be missing. These missing values can then be imputed appropriately.

METHODOLOGY

In this analysis, we need to apply machine learning algorithms that perform the classification task. First, we need to try out different machine learning models on the training dataset itself and then, whichever model performs the best, for different metrics, we need to choose that model. Once the model is chosen, we need to apply the training on the whole training dataset and then based on this, perform predictions on the test dataset. Predictions are made both as class and the probabilities to calculate the various metrics.

Train – test split: Since, there is no external dataset available for the testing of the trained models, we need to split our data into two different datasets - one, that can be used for training the model, and the other that can be used to evaluate the performance of the trained model. About 70% of the data can be used for training the model, and the remaining 30% of the values can be used to test the performance of the model.

Models

After the train – test split has been performed, we need to start training the different machine learning models using the training subset and perform predictions on the test subset. The various machine learning models that are implemented are as follows:

- **Logistic Regression**: This model is an extension of the linear regression model. This model calculates the probability of belonging to a certain class for a data point, based on an equation. The ultimate goal while performing training using this model is to arrive at an equation, based off of which the predictions will be made. It tries to determine various coefficients of an equation. The equation used to calculate the probabilities is:
  $p = 1 / (1 + e^{-logit(p)})$
  where logit(p) is calculated using the formula:
  $logit(p) = a_0 + a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + \ldots$.
  During the training phase, these various coefficients (a0, a1, a2, etc.) are to be determined. And then the logit(p) value is obtained for making a prediction. Then using the formula for p, we calculate p. Thresholds can be set, above or below which, a particular point will lie to different classes.
- **Decision tree model:** A decision tree model works by creating a tree structure and then determining the new values by passing it through this tree structure and finding out the value obtained at the leaf nodes. At each step, information gain or gini index is calculated to determine the best possible column for a split in the decision tree. At the first step, we calculate gini index or information gain to determine which is the best variable to perform the first split. Then, it is done again, to determine the variable for the best split. This process is done until we do cannot make any more splits. Finally, the leaf node is created, by taking the value of the classes that are there for the node before it. So, now, whenever a new value comes, it is passed through this tree structure, and whichever leaf node this data ends up in, is the value of the class for its dependent variable.

- **Random forest:** This model is just like decision trees, except for the fact that instead of creating one single tree, it creates a number of trees, and the final answer is the

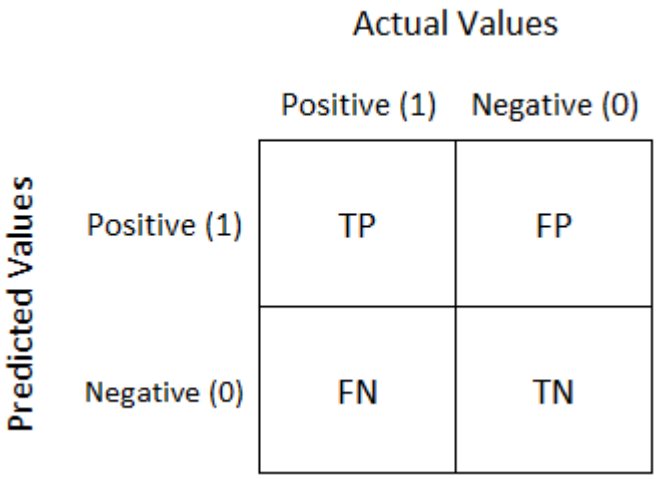
mode of all the answers of the trees. The trees are all different from each other. To create a tree, the model takes up random number of variables and a random subset of the training data. This process is performed for all of the trees. We need to explicitly pass the number of trees that the random forest model should create.

- **Naïve Bayes:** This algorithm works by making use of probabilities of a data point belonging to a certain class. It makes use of the Bayes' theorem in probability which allows us to calculate the probability of a data point belonging to a certain class (P(c|x)). This value is calculated for all the output classes of a data point, and the class for which the probability is the highest, that class is assigned to the data point. Now, to calculate the value of P(c|x), we can make use of Bayes' theorem and calculate the value, P(x|c), for all the variables. The final probability is calculated by using the formula:
  P(c|x) = P(c) * P(x1|c) * P(x2|c) * P(x3|c) * ….
  where P(c) is the probability of the particular class and x1, x2, x3,… are all the independent variables.

CONCLUSION

Model Evaluation: To evaluate the performance of a model, we have various error metrics available. The metrics differ based on the type of task in hand. There are different metrics for evaluating the performance of a classification model and that for a regression model. For this task, which is a classification task, we need to make use of metrics that are appropriate for a classification model. The various metrics that are used in this work are as follows:

- **Confusion Matrix:** Confusion matrix is a matrix that tells the relation between the predicted values and the actual values. In the rows, it contains all the predicted class labels, and in the columns, it contains the actual class labels. In the confusion matrix, it is easy to see whether the predicted value was also the actual value or not. The values which were positive and were predicted to be positive are known as True Positives (TP), while the values that were negative and were predicted to be that are known as True Negatives (TN). The values that were positive and were predicted to be negative are known as False Negatives (FN), while the values that were negative and were predicted to be positive are known as False Positives (FP). The confusion matrix looks as shown below:



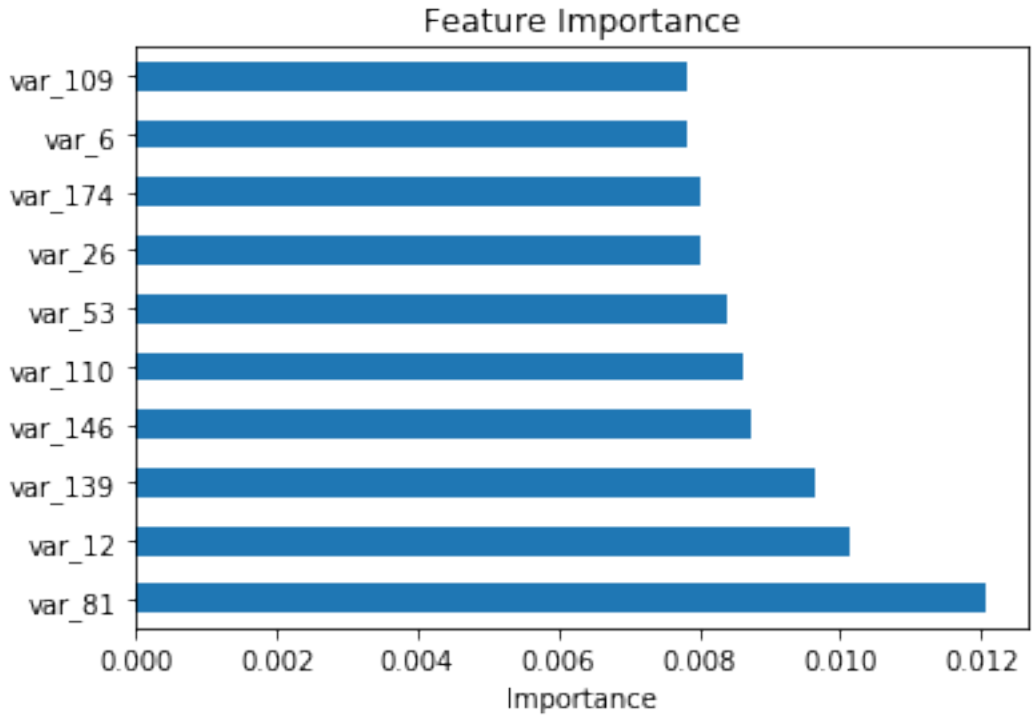
There are various metrics that can be calculated using the confusion matrix. Some of them are as follows:

  - True Positive Rate (TPR)/Recall: It is the ratio of the True Positives with respect to the total positive values.
    TPR = TP / (TP + FN)
  - True Negative Rate (TNR): It is the ratio of the True Negatives with respect to the total negative values.
    TNR = TN / (TN + FP)
  - Precision: It is the ratio of correctly predicted positive values, to the total predicted positive values.
    Precision = TP / (TP + FP)
  - False Positive Rate: It is the ratio of the False Positives to the total negative values that are there.
    FPR = FP / (FP + TN)
  - Accuracy: This is an important performance metric. It tells the accuracy of the model. It tells how many of the predicted values were correct. It is calculated using the formula:
    Accuracy = (TP + TN) / (TP + TN + FP + FN)

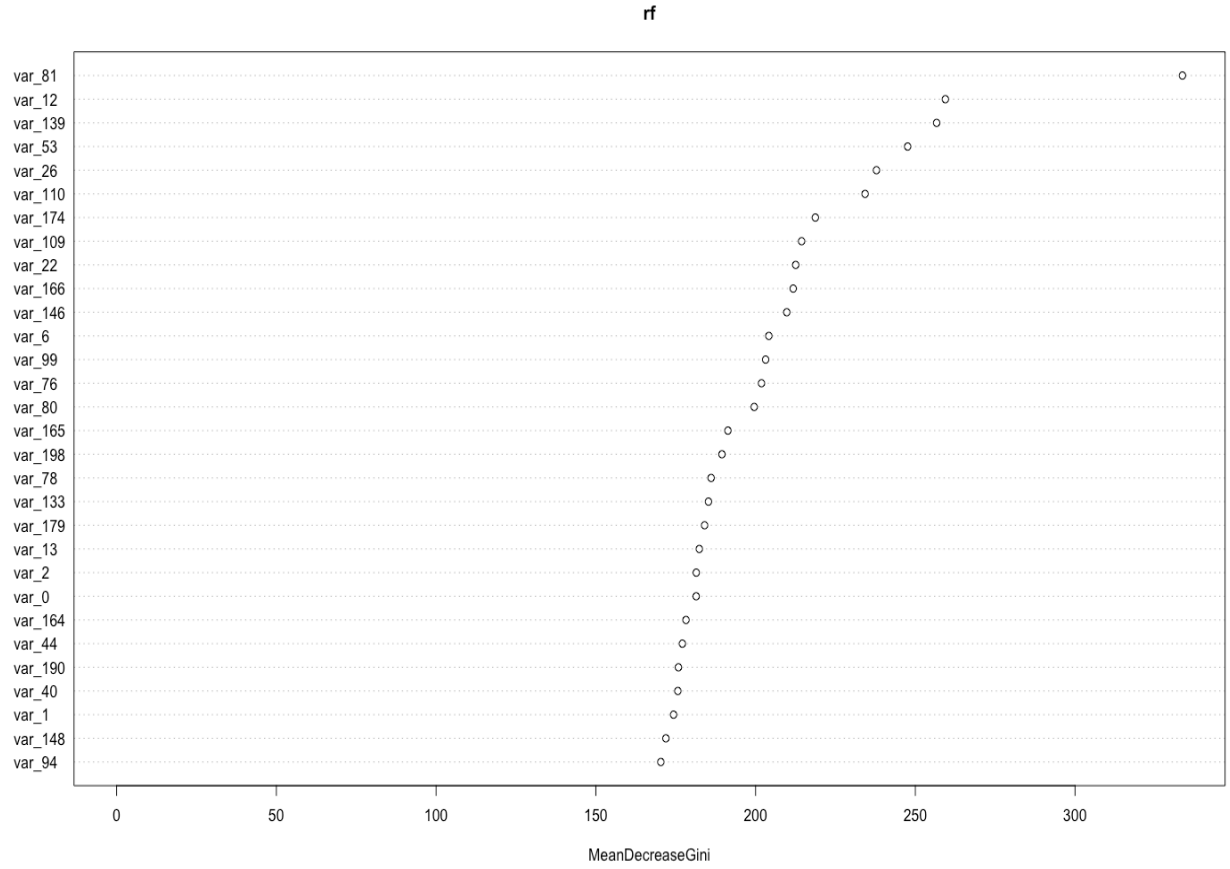
- **AUC:** AUC is the area under the curve of the curve for Receiver Operative Characteristic (ROC). It can be only calculated for binomial classes. The ROC curve is plotted by taking different values of the TPR and the False Positive Rate. To plot the curve, the probabilities calculated by the various models are to be used. Various threshold levels are set, and for that particular threshold level, the value of the TPR and the FPR is determined. The threshold value starts from a threshold of 0 and end at a threshold of 1. The threshold is changed again and again, and accordingly the values of the TPR and the FPR varies. Thus, we can plot these values on a curve. Once the curve is obtained, we can calculate the area under the curve to get the score of AUC for the model.

Variable Importance: Variable importance is used to find out the importance of the various independent variables that are there for the training of the model. It can tell which of the variables were the most important to obtain the classification. Thus, we can reduce the dimensionality of the training dataset by selecting appropriate number of variables for the training. The variable importance can be calculated by making use of the Random Forest model. During the training, whichever variables were mostly selected to perform the splits, are given a higher score than the other variables. Accordingly, for all the trees that it creates, there are scores for the different variables. After the training is complete, we obtain a final score for the importance of the various variables.

Variable importance for the model trained in Python:



Variable importance for the model trained using R:



Model Selection

We can also obtain the performance of the models, using various performance metrics. The performance of the models for the python code is tabulated below:

	AUC	Precision	Recall	Accuracy
Logistic Regression	85.54	26.870	67.782	91.458
Decision Tree	55.60	20.937	19.203	83.412
Random Forest	81.84	0.06	100	90.06
Naïve Bayes	88.53	36.605	70.734	92.19

The performance metrics for the models trained using R is as follows:

	AUC	Precision	Recall	Accuracy
Logistic Regression	86.14	28.14	68.515	91.44
Decision Tree	0.5	0	0	89.91
Random Forest	81.96	0.05	100	89.91
Naïve Bayes	88.82	37.109	71.255	92.14

From the analysis, it is clear that the Naïve Bayes model is performing the best out of all the models. It was also the fastest model to be trained during the training phase. Thus, this model is to be trained once more on the whole training dataset and then can be used to make predictions on the whole test dataset.

APPENDIX

Python code:

```
import pandas as pd

df = pd.read_csv('train.csv')

#Checking the number of null values in the dataset.
pd.set_option('display.max_rows', 202)
df.isnull().sum()

df.head()

#Storing the column names of the numeric columns in a list.
cnames = df.columns[2:]
cnames

arr = []
for i in range(200):
    arr.append(0)

for i in range(200):
    arr[i] = (df.iloc[:,i+2]==0).sum()

test = pd.read_csv('test.csv')

#Checking null values in the test dataset.
test.isnull().sum()

#Checking max values in the dataset to see if there are outliers.
for i in cnames:
    print(df[i].max())

#Checking min values in the dataset to see if there are outliers.
for i in cnames:
    print(df[i].min())

df.dtypes

from sklearn.model_selection import train_test_split

#Train-test split.
x_train, x_test, y_train, y_test = train_test_split(df.iloc[:,2:], df['target'], random_state=1000)

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()

#Implementing Logistic Regression model.
```

```
lrm = lr.fit(x_train, y_train)

#Determining the class labels and the probabily values using the model.
lr_predict = lrm.predict(x_test)
lr_predict_proba = lrm.predict_proba(x_test)

from sklearn.metrics import auc, roc_auc_score, roc_curve, accuracy_score,
confusion_matrix

#AUC score for Logistic Regression model.
roc_auc_score(y_test, lr_predict_proba[:,1])

#Confusion Matrix for the Logistic Regression model.
cm_lr = confusion_matrix(y_test, lr_predict)
cm_lr

#Precision for Logistic Regression model.
precision_lr = cm_lr[1,1] / (cm_lr[1,1] + cm_lr[1,0])
precision_lr

#Recall for Logistic Regression model.
recall_lr = cm_lr[1,1] / (cm_lr[1,1] + cm_lr[0,1])
recall_lr

#Accuracy of the Logistic Regression model.
accuracy_score(y_test, lr_predict)

from sklearn.tree import DecisionTreeClassifier

dtmodel = DecisionTreeClassifier()

#Training the decision tree model.
dt = dtmodel.fit(x_train, y_train)

#Determining the class labels and the probabily values using the model.
dt_predict = dt.predict(x_test)
dt_predict_prob = dt.predict_proba(x_test)

#AUC score the decision tree model
roc_auc_score(y_test, dt_predict_prob[:,1])

#Confusion matrix for decision tree model
cm_dt = confusion_matrix(y_test, dt_predict)
cm_dt

#Precision for the decision tree model
precision_dt = cm_dt[1,1] / (cm_dt[1,1] + cm_dt[1,0])
precision_dt

#Recall for the decision tree model
```

```
recall_dt = cm_dt[1,1] / (cm_dt[1,1] + cm_dt[0,1])
recall_dt

#Accuracy of decision tree model
accuracy_score(y_test, dt_predict)

from sklearn.ensemble import RandomForestClassifier
rfm = RandomForestClassifier(n_estimators = 100)

#Training the Random forest model.
rf = rfm.fit(x_train, y_train)

#Determining the class labels and the probabily values using the model.
rf_predict = rf.predict(x_test)
rf_predict_prob = rf.predict_proba(x_test)

#AUC score for the Random Forest model
roc_auc_score(y_test, rf_predict_prob[:,1])

#Confusion matrix of the Random Forest model
cm_rf = confusion_matrix(y_test, rf_predict)
cm_rf

#Precision for the Random Forest model
precision_rf = cm_rf[1,1] / (cm_rf[1,1] + cm_rf[1,0])
precision_rf

#Recall of the Random Forest model
recall_rf = cm_rf[1,1] / (cm_rf[1,1] + cm_rf[0,1])
recall_rf

#Accuracy of the Random Forest model
accuracy_score(y_test, rf_predict)

from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()

#Training the Naive Bayes model.
nb = gnb.fit(x_train, y_train)

#Determining the class labels and the probabily values using the model.
nb_predict = nb.predict(x_test)
nb_predict_prob = nb.predict_proba(x_test)

#AUC score for the Naive Bayes model.
roc_auc_score(y_test, nb_predict_prob[:,1])

#Confusion matrix for the Naive Bayes model
cm_nb = confusion_matrix(y_test, nb_predict)
cm_nb
```

```
#Precision of the Naive Bayes model
precision_nb = cm_nb[1,1] / (cm_nb[1,1] + cm_nb[1,0])
precision_nb

#Recall of the Naive Bayes model
recall_nb = cm_nb[1,1] / (cm_nb[1,1] + cm_nb[0,1])
recall_nb

#Accuracy of the Naive Bayes model
accuracy_score(y_test, nb_predict)

#Fitting the Naive Bayes model on the whole training dataset
nb_final = gnb.fit(df.iloc[:,2:], df.iloc[:,1])

#Final predictions on the test dataset.
final_predict = nb_final.predict(test.iloc[:,1:])

imp = pd.Series(rf.feature_importances_, index=cnames)

import numpy as np
import matplotlib.pyplot as plt

#Variable importance (Only top 10 variables are shown)
imp.nlargest(10).plot(kind='barh')
plt.title('Feature Importance')
plt.xlabel('Importance')
```

R code:

```
#Installing and importing the necessary libraries.
#install.packages("e1071")
library(e1071)
library(rpart)
#install.packages('randomForest')
library(randomForest)
#install.packages('pROC')
library(pROC)
#nstall.packages('caret')
library(caret)
#Clearing all the unnecessary variables from the environment.
rm(list=ls(all=T))
setwd("/Users/anchitchaturvedi/Desktop/Study/Edwisor/project 3")
getwd()
df = read.csv('train.csv')
head(df)
#Dropping the ID_code column from the dataframe, as it is not required for the analysis.
df = subset(df, select = -c(ID_code) )
#Converting the target column to categorical type.
```

```
df[,1] = sapply(df[,1], as.factor)
head(df)
#Checking null values in the data.
sapply(df, function(x) sum(is.na (x)))
set.seed(100)
#Performing train-test split.
train_index = sample(1:nrow(df), 0.75*nrow(df))
train = df[train_index,]
test = df[-train_index,]
#Implementing the logistic regression model.
logit_model = glm(target ~ ., data = train, family = binomial)
#Getting the probability scores from the model.
lr_prob = predict(logit_model, test[,-1], type = 'response')
lr_prob
#Getting the class labels from the model.
pred_lr = ifelse(lr_prob > 0.5, 1, 0)
#Calculating AUC score for the model.
auc_lr = auc(test[,1], lr_prob)
auc_lr
cf_lr = as.matrix(table(test[,1], pred_lr))
#Confusion Matrix for the Logistic regression model.
confusionMatrix(cf_lr)
#Precision score of the model
precision_lr = cf_lr['1','1'] / (cf_lr['1','1'] + cf_lr['1','0'])
precision_lr
#Recall of the model
recall_lr = cf_lr['1','1'] / (cf_lr['1','1'] + cf_lr['0','1'])
recall_lr
#Training the decision tree model
dt = rpart(target ~., train)
#Getting class labels using the model
pred_dt = predict(dt, test[,-1], type = 'class')
pred_dt
#Getting the probability scores using the model.
dt_prob = predict(dt, test[,-1], type = 'prob')
#AUC score of the model
auc_dt = auc(test[,1], dt_prob[,2])
auc_dt
#Confusion Matrix of the model
cf_dt = as.matrix(table(test[,1], pred_dt))
confusionMatrix(cf_rf)
#Precision score for the model
precision_dt = cf_dt['1','1'] / (cf_dt['1','1'] + cf_dt['1','0'])
precision_dt
#Recall of the decision tree model.
recall_dt = cf_dt['1','1'] / (cf_dt['1','1'] + cf_dt['0','1'])
recall_dt
#Implementing the Naive Bayes algorithm
nb = naiveBayes(target ~., train)
#Predicting class labels using the trained model
```

```
pred_nb = predict(nb, test[,-1], type = 'class')
pred_nb
#Predicting probability scores using the model.
nb_prob = predict(nb, test[,-1], type = 'raw')
nb_prob
#AUC score for the model
auc_nb = auc(test[,1], nb_prob[,2], levels = c(0, 1), direction = "<")
auc_nb
#Confusion matrix of the Naive Bayes model
cf_nb = as.matrix(table(test[,1], pred_nb))
confusionMatrix(cf_nb)
#Precision score for the model
precision_nb = cf_nb['1','1'] / (cf_nb['1','1'] + cf_nb['1','0'])
precision_nb
#Recall for the model
recall_nb = cf_nb['1','1'] / (cf_nb['1','1'] + cf_nb['0','1'])
recall_nb
#Training the Random Forest model
set.seed(100)
rf = randomForest(target ~., train, ntree=100, importance= TRUE)
#Predicting class labels using the trained model.
pred_rf = predict(rf, test[,-1], type = 'class')
rf
#Getting probability scores using the trained model
rf_prob = predict(rf, test[,-1], type = 'prob')
rf_prob
#AUC for the model
auc_rf = auc(test[,1], rf_prob[,2])
auc_rf
#Confusion matrix for the Random forest model.
cf_rf = as.matrix(table(test[,1], pred_rf))
confusionMatrix(cf_rf)
#Precision score of the model
precision_rf = cf_rf['1','1'] / (cf_rf['1','1'] + cf_rf['1','0'])
precision_rf
#Recall score of the model
recall_rf = cf_rf['1','1'] / (cf_rf['1','1'] + cf_rf['0','1'])
recall_rf
#Variable Importance using the Random Forest model
varImpPlot(rf, type = '2')
test_set = read.csv('test.csv')
#Training the Naive Bayes model on the whole training dataset
nbfinal = naiveBayes(target ~., df)
#Performing final predictions on the test dataset using the model
final_pred = predict(nbfinal, test_set[,-1], type = 'class')
```