Santander Transaction Customer Prediction

PRUDHVI KUMAR K

15th October, 2019

Contents

Appendix A - Extra

Chapter-1 Introduction

1.1 Problem Statement

About Santander - 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.

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

1.2 Data

The details of data set is given below.

The data is comprised in two partitions.

The Train.csv

The Test.csv

The train data set comprised of 200000 observations and 202 features.

In which there are 200 numerical features, one factor (string) ID_code and one dependant feature -target.

The test data set comprised of 200000 observations and 201 features.

In which there are 200 numerical features and string ID_code,

We need to predict the target variable in the test data set.

Chapter 2

Methodology

2.1 .1 Pre Processing

The work starts with data preprocessing, means looking at the data to get insights. However, in data mining terms *looking at data* refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as **Exploratory Data Analysis**.

To start this process we will first try and look at all the distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the distributions of the variable by QQ-normality graph or histogram. Histogram is the best chart to represent the data distribution, later the data is normalized or standardized, according to the model needs.

And we check for distribution of the target variable with respect to its independent features.

And what if we found imbalanced dataset ?

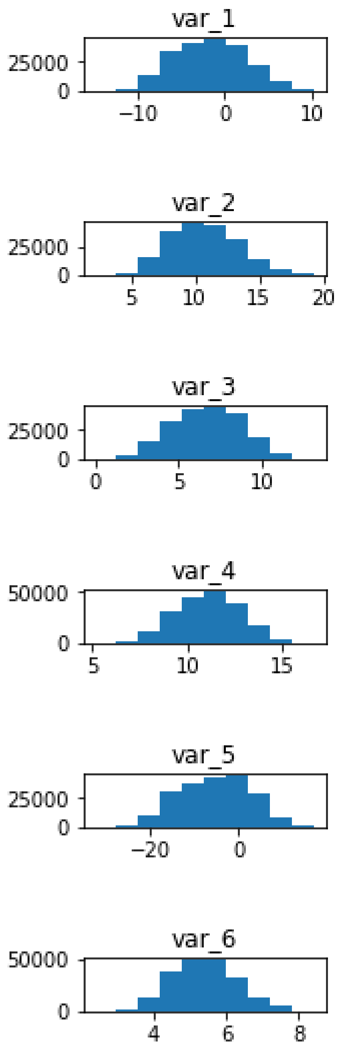
Data preprocessing is the tedious task, we need to focus more on this part to reduce the model complexity. Before feeding the data to model, we must preprocess the data, which has various stages like missing value analysis, impute the missing values, outlier check, normality check, sampling, weather the dataset is imbalanced are not. Then the data is subjected to model training and prediction.

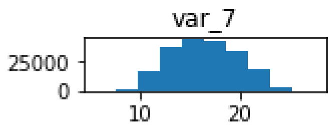
After completing the model prediction, the machine learning prediction system is ready for deployment. Project deployment refers to , preparing the machine learning environment to the front end users.

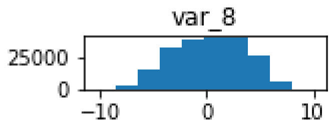
Distribution of the features.

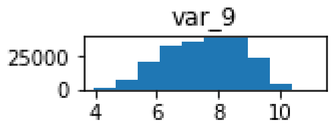
So checking the distribution of the features available in the train dataset, the below graphs are the indication that all the features comprised of normalized distribution.

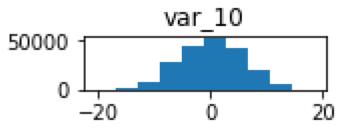
The same distribution is observed in the test data set.

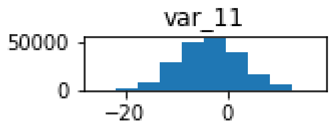


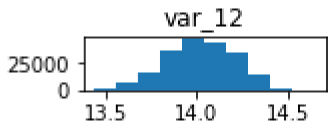

var_7
25000
0
10
20

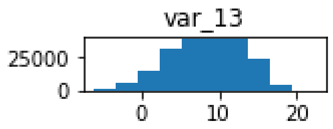

var_8
25000
0
−10
0
10

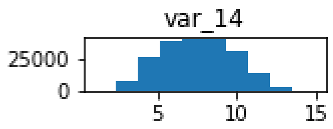

var_9
25000
0
4
6
8
10

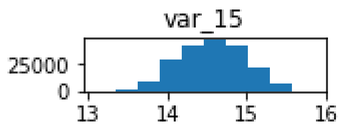

var_10
50000
0
−20
0
20


var_11
50000
0
−20
0

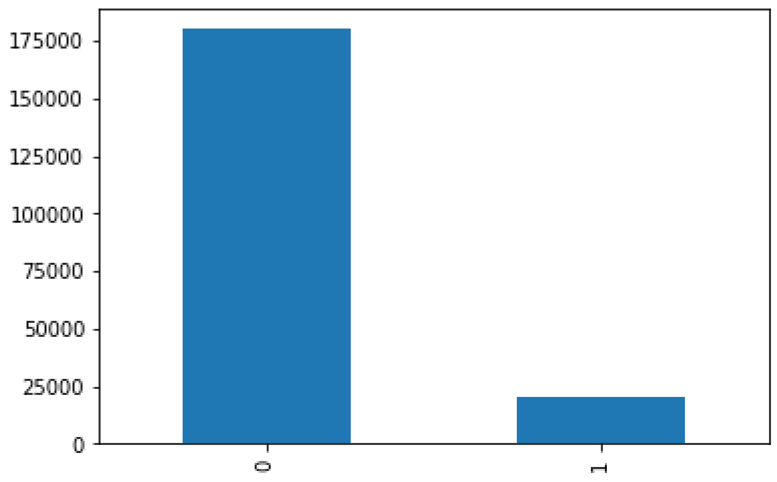

var_12
25000
0
13.5
14.0
14.5


var_13
25000
0
0
10
20


var_14
25000
0
5
10
15


var_15
25000
0
13
14
15
16

Target variable distribution.



The target variable is comprised of two categories.

0 and 1. Respectively!

0 is like the customer , not made a transaction.

1 is like the customer made a transaction.

Here in our case , got to observe an Imbalanced target variable.

#0=179902

# 1=20098

Almost 80% of target feature is dominated by 0.

How to handle imbalanced target class feature.

Using the appropriate evaluation metrics.

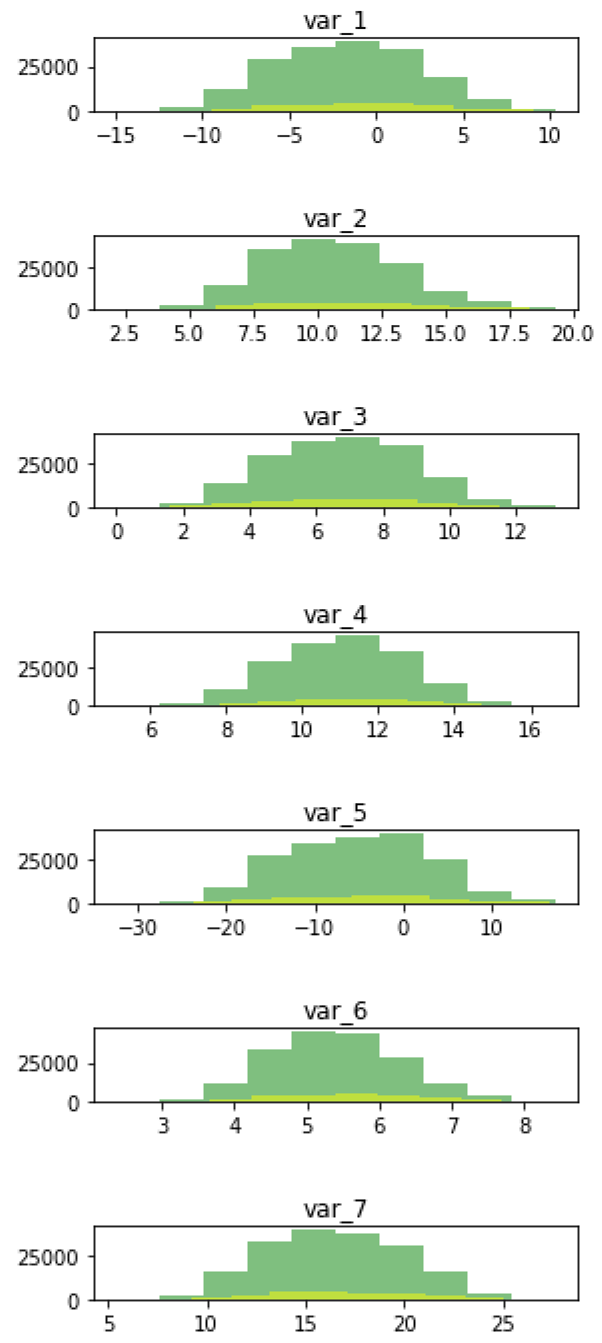
Using the appropriate algorithm.

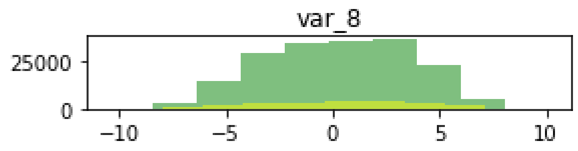
Using the appropriate sampling method.

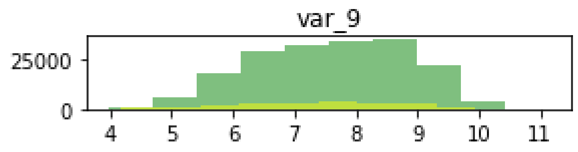
Let's check the distribution of target variable with respect to its independent features.

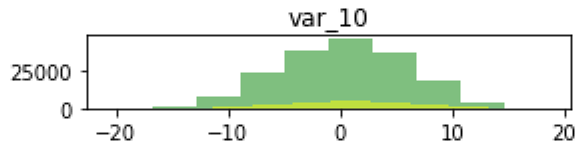
This can be done either violin plots are histogram.

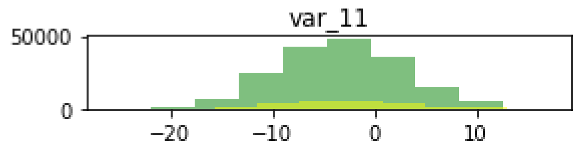
NOTE:- Light green parts are 0 and at bottom yellow color are 1.

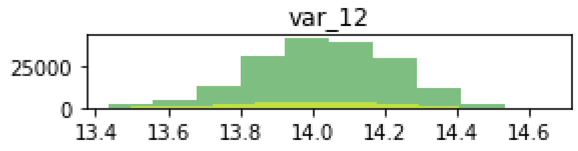


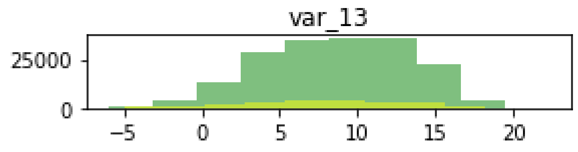

var_8
25000
0
−10
−5
0
5
10

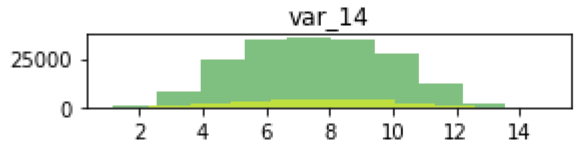

var_9
25000
0
4
5
6
7
8
9
10
11

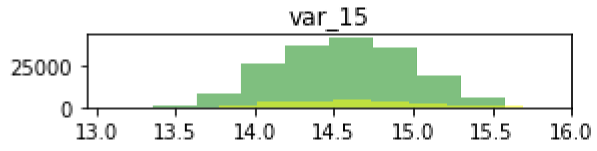

var_10
25000
0
−20
−10
0
10
20


var_11
50000
0
−20
−10
0
10


var_12
25000
0
13.4
13.6
13.8
14.0
14.2
14.4
14.6


var_13
25000
0
−5
0
5
10
15
20


var_14
25000
0
2
4
6
8
10
12
14


var_15
25000
0
13.0
13.5
14.0
14.5
15.0
15.5
16.0

Observation

- There are no missing values in the Train and Test Data sets.
- There are outliers present in all features.(Both Train and Test data)
- There are no categorical Features in the data sets, except the string ID_code.
- The features are not skewed. (Both Train and Test data)
- The features show the normal distribution. (Both Train and Test data.)
- The target feature is imbalanced .(Applies to train data)
- There is the minimal distribution of class 1 target feature in all independent features. (Applies to train data only)
- There is the max distribution of class 0 target feature in all independent features.(Applies to train data only)
- The target feature of the test data set must be predicted.
- Finally the algorithm suitable for this problem is CLASSIFICATION type algorithm.

Few of the classification Algorithm are:-

- Logistic regression.
- Decision tree classifier.
- Random forest classifier.
- Naive bayes classifier.

2.1.2 Feature Engineering.

The term Feature Engineering refers to transforming the features available in the dataset.

Transforming in sense, preparing the features ready to use for model building . This includes

- How to handle missing values in the data set ?
- How to handle outliers in the data set ?
- How to handle numerical features in the data set ?.
- How to handle categorical features in the data set ?
- How to handle rare categorical labels in the data set ?
- How to handle skewed features, we found in pre processing ?

Missing values.

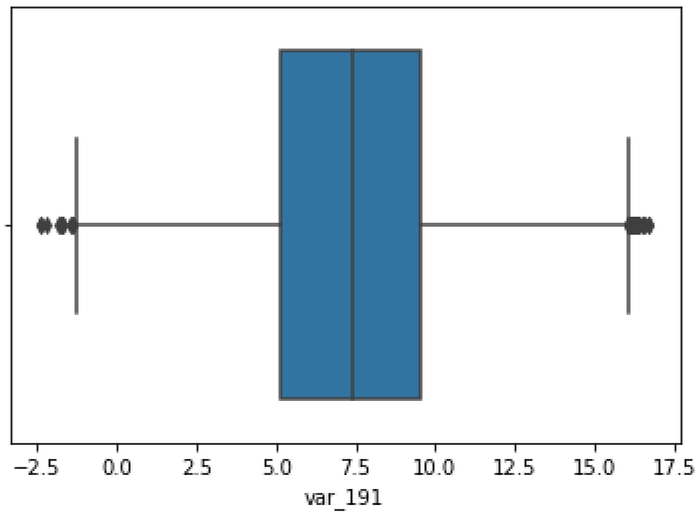
During the pre processing of the santander transaction data, from observation one can conclude that there are no missing values in the data sets.

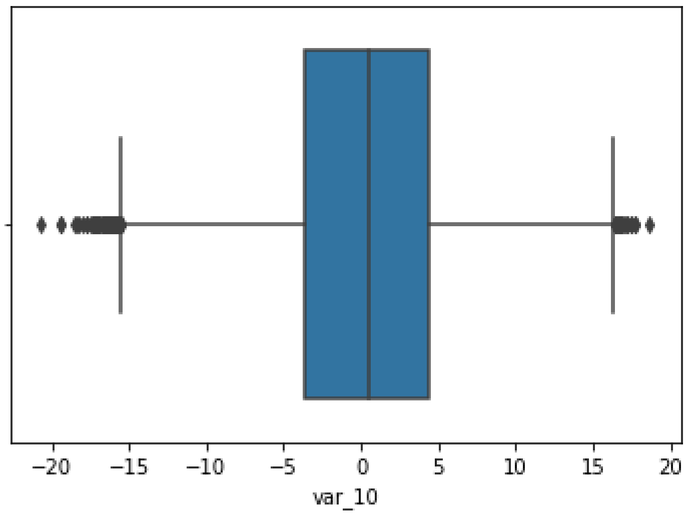
Outliers

There are lots of outliers in both and train and test datasets.

For this problem we are heading with outliers, as they are found almost 90% of the train and test data sets.

If imputation is done, like we are changing almost all the data points.which may in model building predicts false outcomes.





Handling numerical Features in the data set

The features are not skewed , all the features fall under normal distribution.

The real challenge is to choose the best model and best evaluation metrics for this imbalanced target class data set. We need to predict the target feature of the test data set.

2.1.3 Feature selection

Before performing any type of modelling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction.

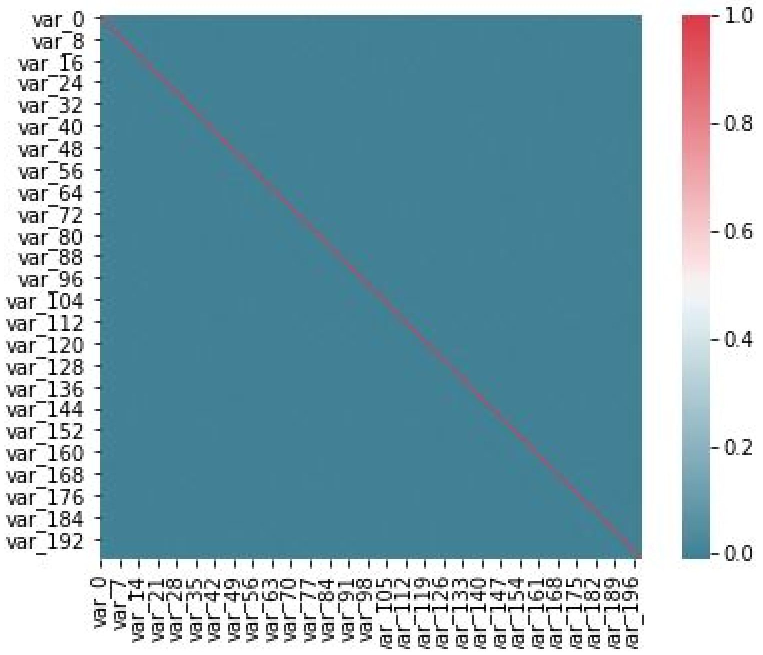
For all the numerical variable present in the training data set, there are 200 numerical features.

The objective of this selection is to neglect the features with high correlation.

There are several methods of doing that. Below we have used the **correlation plot for checking multicollinearity.**

Here we use correlation plot.

The best part is , there is a usage of eli5 library to show the feature importance and its weights ,explaining which features are affected or contributing our model.(In appendix part)



The figure above shows that most of the pearson correlations between the numerical features are close to Zero. That means most of the numerical features are almost uncorrelated between them.

Chapter 3

3.1 Data Modelling

3.1.1 Model selection

Based on the target variable or feature we select the model. Here in our case in Santander Transaction customer prediction data set, the target variable is comprised of binary classification., so the model will be the classification model.

Classification Model

Classification is a supervised **machine learning** method. It always requires labeled training data. When training is finished, you can evaluate and tune the **model**. When you're satisfied with the **model**, use the trained **model** for scoring with new data.

The classification technique is a systematic approach to build classification models from an input data set. For example, decision tree classifiers, rule-based classifiers, neural networks, support vector machines, and naive Bayes classifiers are different techniques to solve a classification problem. Each technique adopts a learning algorithm to identify a model that best fits the relationship between the attribute set and class label of the input data. Therefore, a key objective of the learning algorithm is to build a predictive model that accurately predict the class labels of previously unknown records.

3.1.2 Logistic regression

In statistics, the logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc... Each object being detected in the image would be assigned a probability between 0 and 1 and the sum adding to one.

```
Call:
glm(formula = target ~ ., data = train1)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.60425  -0.15020  -0.06576   0.03069   1.15669

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.949e+00  5.214e-01  11.409  < 2e-16 ***
var_0        4.109e-03  2.393e-04  17.174  < 2e-16 ***
var_1        2.982e-03  1.798e-04  16.589  < 2e-16 ***
var_2        5.395e-03  2.754e-04  19.592  < 2e-16 ***
var_3        1.775e-03  3.562e-04   4.983 6.27e-07 ***
var_4        1.804e-03  4.483e-04   4.023 5.74e-05 ***
var_5        1.095e-03  9.257e-05  11.827  < 2e-16 ***
var_6        1.952e-02  8.392e-04  23.256  < 2e-16 ***
var_7       -2.681e-04  2.127e-04  -1.261 0.207399
var_8        1.408e-03  2.179e-04   6.459 1.06e-10 ***
var_9       -8.361e-03  5.894e-04 -14.184  < 2e-16 ***
var_10      -5.050e-06  1.323e-04  -0.038 0.969550
```

Rest of the features are omitted.

Logit model

```
Call:  glm(formula = target ~ ., data = train1)
```

```
Coefficients:
(Intercept)         var_0         var_1         var_2
var_3         var_4         var_5         var_6
  5.949e+00    4.109e-03    2.982e-03    5.395e-03
1.775e-03    1.804e-03    1.095e-03    1.952e-02
      var_7         var_8         var_9        var_10
var_11        var_12        var_13        var_14
 -2.681e-04    1.408e-03   -8.361e-03   -5.050e-06
9.215e-04   -9.062e-02   -3.109e-03   -5.992e-04
```

Prediction

```
 1             2             3             4             5
6             7
 0.2241730130  0.2497485689  0.1139615033  0.2372842824
0.1327701987 -0.1220606825 -0.0273597794
```

3.1.3 Decision Tree Classifier

A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursive manner call recursive partitioning. This flowchart-like structure helps you in decision making.

Decision Tree Classifier is a simple and widely used classification technique. It applies a straightforward idea to solve the classification problem. Decision Tree Classifier poses a series of carefully crafted questions about the attributes of the test record. Each time it receive an answer, a follow-up question is asked until a conclusion about the class label of the record is reached.

```
Call:
C5.0.formula(formula = target ~ ., data = train1, trials =
3, rules = TRUE)


C5.0 [Release 2.07 GPL Edition]      Tue Oct 15 14:24:20
2019
-------------------------------

Class specified by attribute `outcome'

Read 160000 cases (201 attributes) from undefined.data

-----  Trial 0:  -----

Rules:

Rule 0/1: (31602/1072, lift 1.1)
     var_2 <= 11.3653
     var_13 > 0.3237
     var_22 <= 8.5108
     var_40 <= 6.9101
     var_53 <= 7.2852
     var_75 > 5.91
     var_78 <= 8.7111
     var_80 > -7.735
```

Predictions

```
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
```

3.1.4 Random Forest Classifier

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

```
    condition
          [1,]   "var_12<=1.78475   &   var_79<=-5.6936   &
var_106<=16.6236  &  var_108<=10.27155  &  var_162<=8.23235  &
var_165<=3.293"
          [2,]   "var_12<=1.78475   &   var_79<=-5.6936   &
var_106<=16.6236  &  var_108<=10.27155  &  var_162>8.23235  &
var_165<=3.293"
```

Predictions

```
639  640  641  642  643  644  645  646  647  648  649  650
651  652  653  654  655  656  657  658  659  660
   0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0
```

3.1.5 Naive Bayes

Bayes' theorem is a theorem used to calculate the probability of something being true, false, or a certain way. Bayes' theorem is an extension of logic. It expresses how a belief should change to account for evidence.

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

As it consists of real time learning about the features , For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features. this is considered as the best way to deal with target class imbalance datasets, if we fail to collect extra features that can balance the data and fail to resample the data.

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
         0          1 
0.90000625 0.09999375 
```

```
Conditional probabilities:
   var_0
Y        [,1]     [,2]
  0 10.62616 3.004929
  1 11.13959 3.282909

   var_1
Y          [,1]     [,2]
  0 -1.6911799 4.027634
  1 -0.9998758 4.232729

   var_2
Y        [,1]     [,2]
  0 10.66584 2.613247
  1 11.16563 2.853125
```

Predictions

```
 1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  [53] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [105] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Chapter 4

Conclusion

4.1.1Model Evaluation

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the models using any of the following criteria:

1. Predictive Performance
2. Interpretability
3. Computational Efficiency

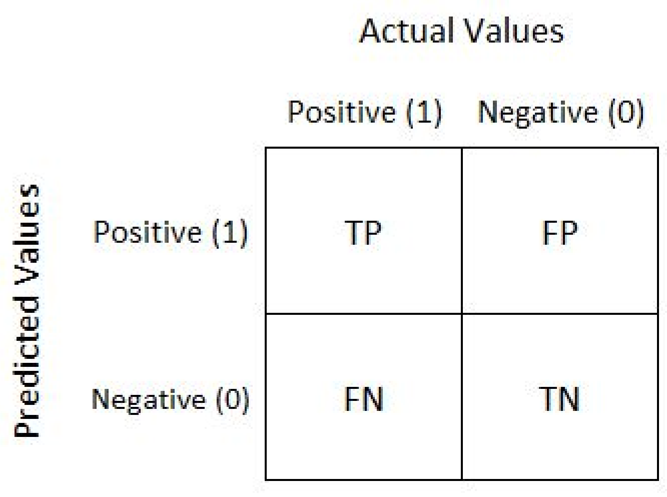
Evaluating a model is a core part of building an effective machine learning model There are several evaluation metrics, like confusion matrix, cross-validation, AUC-ROC curve, etc. Different evaluation metrics are used for different kinds of problems.

4.1.2 Confusion Matrix

Here we build a confusion matrix with help of TRAIN dataset, which is provided. NOTE:- The train data set is splitted into two parts for training and testing, for evaluating the model. As the test data set provided with no target variable , we cannot use it for confusion matrix building.

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

It is a table with 4 different combinations of predicted and actual values.



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

True Positive:

Interpretation: You predicted positive and it’s true.predicted that a customer transaction is done and actually it is.

True Negative:

Interpretation: You predicted negative and it’s true.predicted that a customer transaction is not done and its actually not.

False Positive: (Type 1 Error)

Interpretation: You predicted positive and it’s false.predicted that a customer transaction is done, but it's actually not.

False Negative: (Type 2 Error)

Interpretation: You predicted negative and it’s false.predicted that customer transaction is not made but actually it is.

We consider predicted values as Positive and Negative and actual values as True and False.

Formula for Calculating Accuracy

Accuracy= (TP+TN)*100/(TP+TN+FP+FN)

Formula for Calculating FNR

FNR= (FN*100)/(FN+TP)

False negative rate must be low as possible for any model. Since it is the prediction that transaction is not made and actually it’s made.

Formula for Calculating Recall

RECALL= (TP*100)/(TP+FN)

Out of all the positive classes, how much we predicted correctly. It should be high as possible.

Formula for Calculating Precision

PRECISION= (TP*100)/(TP+FP)

Out of all the positive classes we have predicted correctly, how many are actually positive.

Formula for calculating F score

F Score= (2*RECALL*PRECISION)/(RECALL+PRECISION)

It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

Confusion matrix of Decision tree.

```
Confusion Matrix and Statistics

   tree_predictions
        0     1
  0 35368   533
  1  3716   383

              Accuracy : 0.8938
                95% CI : (0.8907, 0.8968)
   No Information Rate : 0.9771
   P-Value [Acc > NIR] : 1

                 Kappa : 0.1198

 Mcnemar's Test P-Value : <2e-16
```

```
            Sensitivity : 0.90492
            Specificity : 0.41812
         Pos Pred Value : 0.98515
         Neg Pred Value : 0.09344
             Prevalence : 0.97710
         Detection Rate : 0.88420
   Detection Prevalence : 0.89753
      Balanced Accuracy : 0.66152

       'Positive' Class : 0
```

Accuracy=89.38 %

FNR= 9.50 %

Recall= 90.4 %

precision= 98.4 %

F score= 94.1 %

Confusion matrix of Random Forest.

```
Confusion Matrix and Statistics

   rf_Predictions
        0     1
  0 35900     1
  1  4097     2

               Accuracy : 0.8976
                 95% CI : (0.8945, 0.9005)
    No Information Rate : 0.9999
    P-Value [Acc > NIR] : 1
```

```
                  Kappa : 8e-04

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.8975673
            Specificity : 0.6666667
         Pos Pred Value : 0.9999721
         Neg Pred Value : 0.0004879
             Prevalence : 0.9999250
         Detection Rate : 0.8975000
   Detection Prevalence : 0.8975250
      Balanced Accuracy : 0.7821170

       'Positive' Class : 0
```

Accuracy=89.76 %

FNR= 10.2 %

Recall= 89.7 %

Precision= 99.9 %

F score= 94.5 %

Confusion matrix of Naive Bayes.

```
Confusion Matrix and Statistics

   nb_predicion
```

```
        0     1
  0 35341   560
  1  2585  1514

               Accuracy : 0.9214
                 95% CI : (0.9187, 0.924)
    No Information Rate : 0.9482
    P-Value [Acc > NIR] : 1

                  Kappa : 0.4528

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9318
            Specificity : 0.7300
         Pos Pred Value : 0.9844
         Neg Pred Value : 0.3694
             Prevalence : 0.9482
         Detection Rate : 0.8835
   Detection Prevalence : 0.8975
      Balanced Accuracy : 0.8309
                                       'Positive' Class : 0
```

Accuracy= 92.1 %

FNR= 6.8 %

Recall= 93.1 %

precision= 98.4 %

F score= 95.6 %

4.1.3 Observations

As we can see the Naive Bayes classifier has the best accuracy(92.1%) and low FNR(6.8%).

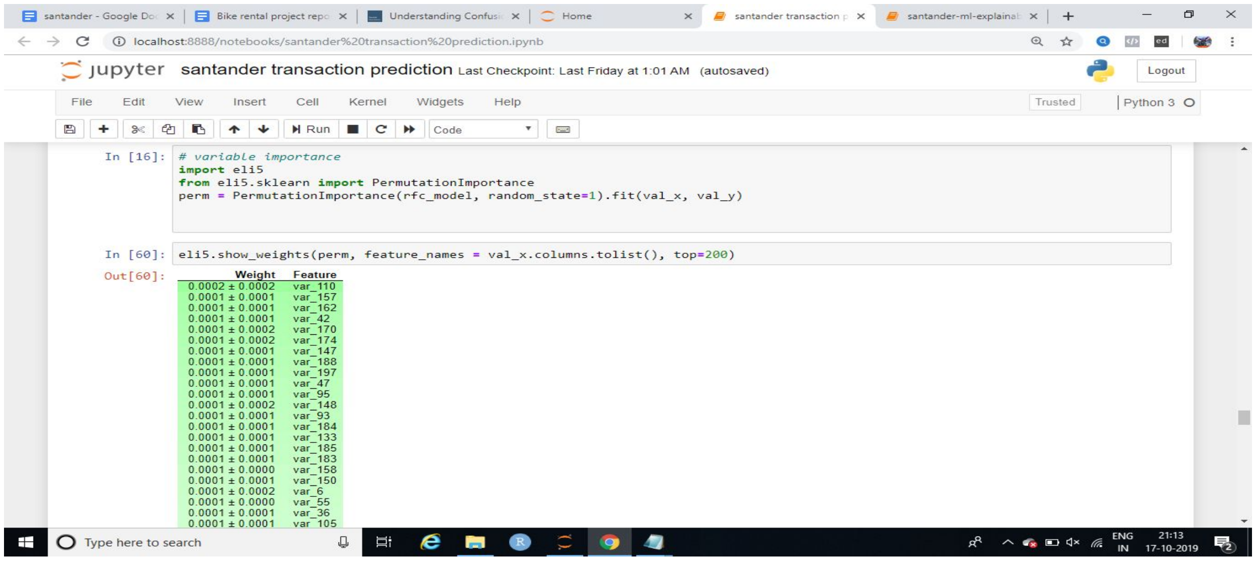
Here by we conclude , and freeze naive bayes classifier as the best fit for this santander customer transaction prediction.

WHY NAIVE BAYES ?

As it consists of real time learning about the features , For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features. this is considered as the best way to deal with target class imbalance datasets, if we fail to collect extra features that can balance the data and fail to resample the data.

Here we cannot access the extra features to balance the dataset as there is target class imbalance . And cannot be resampled as there is a risk of changing the information of data available.

Appendix A - Variable/ Feature importance.



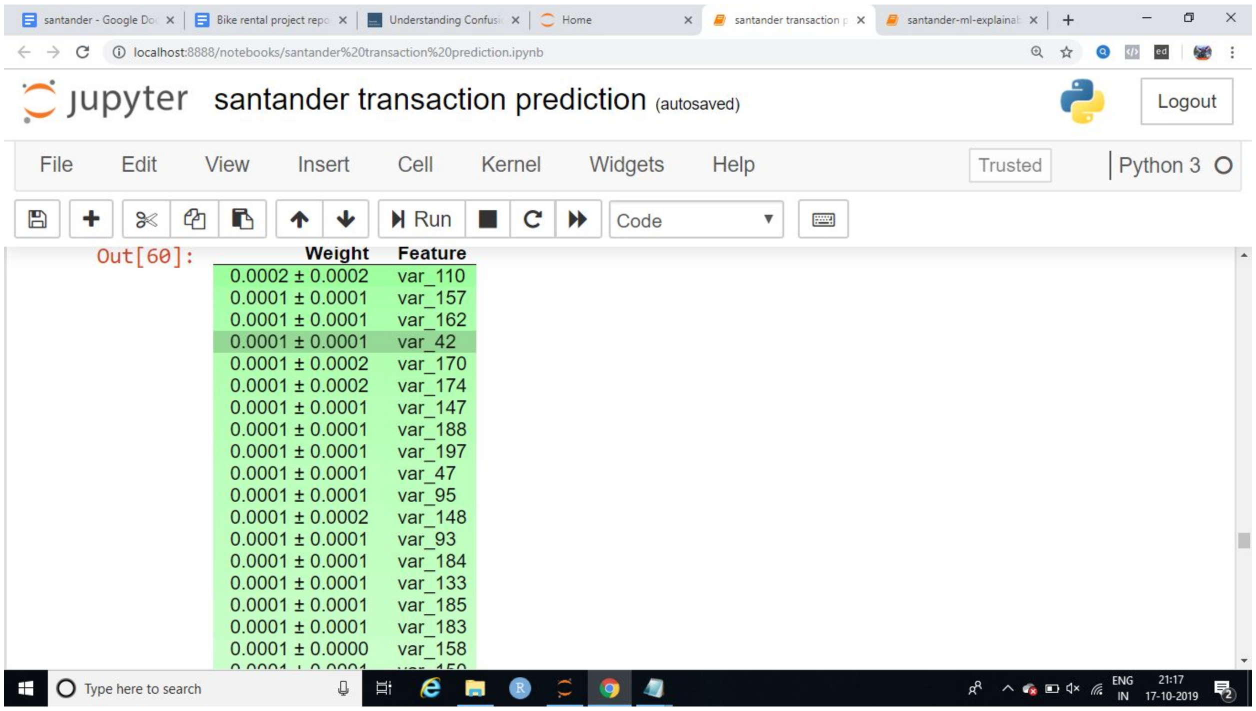
```
In [16]: # variable importance
         import eli5
         from eli5.sklearn import PermutationImportance
         perm = PermutationImportance(rfc_model, random_state=1).fit(val_x, val_y)

In [60]: eli5.show_weights(perm, feature_names = val_x.columns.tolist(), top=200)
```

Out[60]:

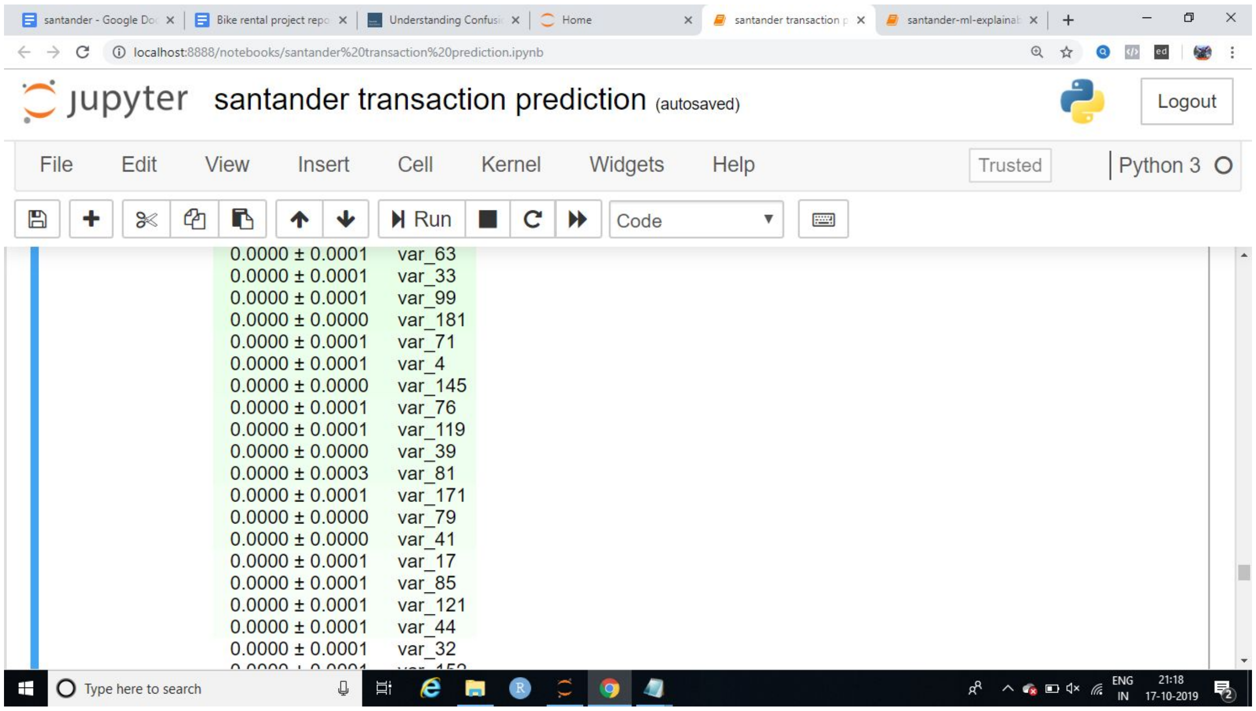
Weight	Feature
0.0002 ± 0.0002	var_110
0.0001 ± 0.0001	var_157
0.0001 ± 0.0001	var_162
0.0001 ± 0.0001	var_42
0.0001 ± 0.0002	var_170
0.0001 ± 0.0002	var_174
0.0001 ± 0.0001	var_147
0.0001 ± 0.0001	var_188
0.0001 ± 0.0001	var_197
0.0001 ± 0.0001	var_47
0.0001 ± 0.0001	var_95
0.0001 ± 0.0002	var_148
0.0001 ± 0.0001	var_93
0.0001 ± 0.0001	var_184
0.0001 ± 0.0001	var_133
0.0001 ± 0.0001	var_185
0.0001 ± 0.0001	var_183
0.0001 ± 0.0000	var_158
0.0001 ± 0.0001	var_150
0.0001 ± 0.0002	var_6
0.0001 ± 0.0000	var_55
0.0001 ± 0.0001	var_36
0.0001 ± 0.0001	var_105

Eli5 library is used to debug the classifiers, and explain their predictions.

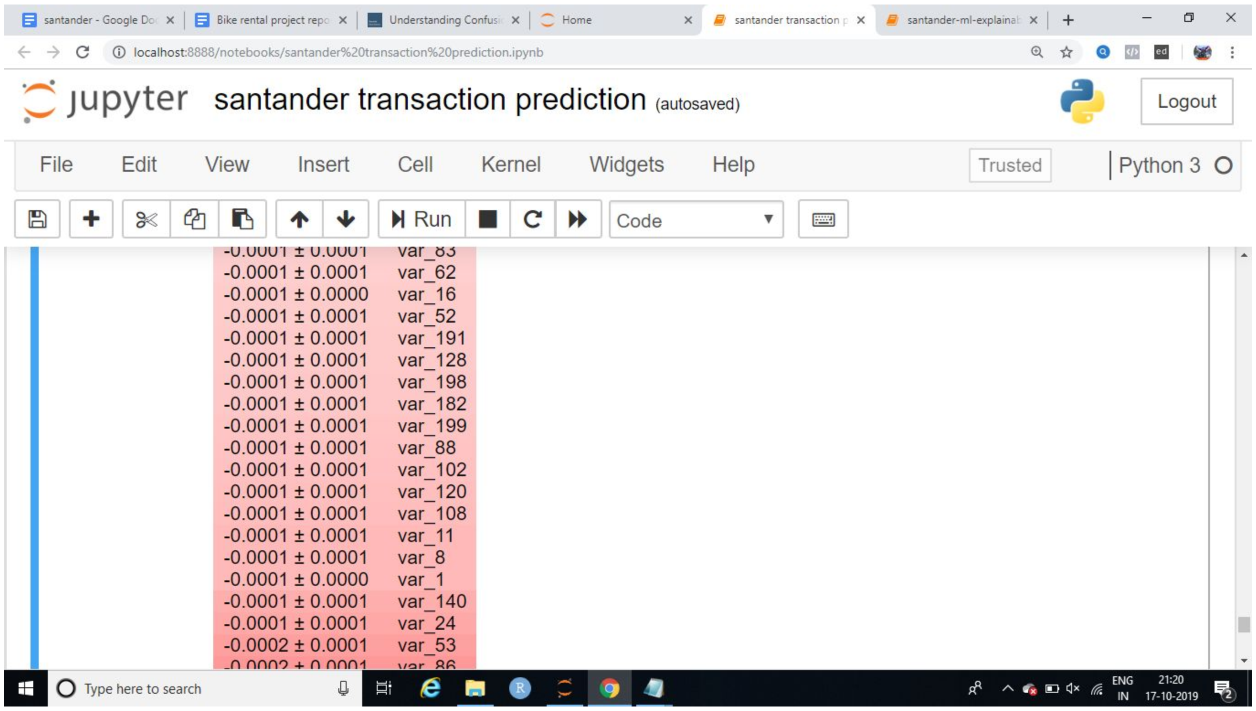


Out[60]:

Weight	Feature
0.0002 ± 0.0002	var_110
0.0001 ± 0.0001	var_157
0.0001 ± 0.0001	var_162
0.0001 ± 0.0001	var_42
0.0001 ± 0.0002	var_170
0.0001 ± 0.0002	var_174
0.0001 ± 0.0001	var_147
0.0001 ± 0.0001	var_188
0.0001 ± 0.0001	var_197
0.0001 ± 0.0001	var_47
0.0001 ± 0.0001	var_95
0.0001 ± 0.0002	var_148
0.0001 ± 0.0001	var_93
0.0001 ± 0.0001	var_184
0.0001 ± 0.0001	var_133
0.0001 ± 0.0001	var_185
0.0001 ± 0.0001	var_183
0.0001 ± 0.0000	var_158



```
0.0000 ± 0.0001   var_63
0.0000 ± 0.0001   var_33
0.0000 ± 0.0001   var_99
0.0000 ± 0.0000   var_181
0.0000 ± 0.0001   var_71
0.0000 ± 0.0001   var_4
0.0000 ± 0.0000   var_145
0.0000 ± 0.0001   var_76
0.0000 ± 0.0001   var_119
0.0000 ± 0.0000   var_39
0.0000 ± 0.0003   var_81
0.0000 ± 0.0001   var_171
0.0000 ± 0.0000   var_79
0.0000 ± 0.0000   var_41
0.0000 ± 0.0001   var_17
0.0000 ± 0.0001   var_85
0.0000 ± 0.0001   var_121
0.0000 ± 0.0001   var_44
0.0000 ± 0.0001   var_32
0.0000 ± 0.0001   var_152
```



```
-0.0001 ± 0.0001   var_83
-0.0001 ± 0.0001   var_62
-0.0001 ± 0.0000   var_16
-0.0001 ± 0.0001   var_52
-0.0001 ± 0.0001   var_191
-0.0001 ± 0.0001   var_128
-0.0001 ± 0.0001   var_198
-0.0001 ± 0.0001   var_182
-0.0001 ± 0.0001   var_199
-0.0001 ± 0.0001   var_88
-0.0001 ± 0.0001   var_102
-0.0001 ± 0.0001   var_120
-0.0001 ± 0.0001   var_108
-0.0001 ± 0.0001   var_11
-0.0001 ± 0.0001   var_8
-0.0001 ± 0.0000   var_1
-0.0001 ± 0.0001   var_140
-0.0001 ± 0.0001   var_24
-0.0002 ± 0.0001   var_53
-0.0002 ± 0.0001   var_86
```

What can be inferred from the above?

1. As you move down the top of the graph, the importance of the feature decreases.
2. The features that are shown in green indicate that they have a positive impact on our prediction
3. The features that are shown in white indicate that they have no effect on our prediction
4. The features shown in red indicate that they have a negative impact on our prediction
5. The most important feature was **Var_110**.

NOTE :- For full explanation check jupyter notebook attached.

Complete R File

```
rm(list=ls())
# setting the working directory.
setwd("D:/R and PYTHON files/data set/project 2")
getwd()

# loading the train and test data to the environment.
train=read.csv("train.csv",header=TRUE)
test=read.csv("test.csv",header=TRUE)

# installing the necessary libraries.
#install.packages (c("ggplot2", "corrgram", "DMwR",
"caret", "randomForest", "unbalanced", "C50", "dummies",
"e1071", "Information",
                    #"MASS", "rpart", "gbm", "ROSE",
'sampling', 'DataCombine', 'inTrees'))

#  exploring the datasets
dim(train)# 200000 observation and 202 features
dim(test) # 200000 observations and 201 features
str(train) # to know the features class type
str(test) # to know the features class type

# checking for missing values.
sum(is.na(train))# no missing values in train dataset
sum(is.na(test)) # no missing values in test dataset

# target feature analysis.
unique(train$target)# unique values are 0 and 1.
table(train$target)# 0=179902, 1=20098
hist(train$target)# the train data has one dependent
```

variable called "target" whose values are compromised of imbalance class values.

```
# visualization
library(ggplot2)
require(gridExtra)
# barplot
plot=ggplot(train,aes(target))+theme_bw()+geom_bar(stat='co
unt',fill='brown')
grid.arrange(plot)
```

```
# deleting the features which are not wanting.
train= subset(train,select=-c(ID_code))
test= subset(test,select=-c(ID_code))
```

```
# preparing the data for model developement
library(caret)
train_index=createDataPartition(train$target, p = .80, list
= FALSE)
train1=train[train_index,]
test2=train[-train_index,]
```

```
# for logistic regression target variable must be numeric.
train1$target=as.numeric(train1$target)
# logistic regression modell
logit_model=glm(target~.,data=train1)
```

```
summary(logit_model)
logit_predictions=predict(logit_model,newdata=test2[,-1],ty
pe="response")
# predicting for test data
test_predictions=predict(logit_model,newdata=test,type="res
ponse")
#write(capture.output(test_predictions),"logistic model
predictions new R.txt")

# decision tree classifier.
library(C50)
# tree model requires a factor target variable
train1$target=as.factor(train1$target)
tree_model= C5.0(target ~., train1, trials = 3, rules =
TRUE)
tree_predictions=predict(tree_model,test2[,-1],type="class"
)
# prediction of test case dependent feature.
testcase_pred=predict(tree_model,test,type="class")
# model evaluation
ConfMatrix_tree_mat = table(test2$target,tree_predictions )
confusionMatrix(ConfMatrix_tree_mat)
# Accuracy=89.38 %
# FNR=  9.50 %
# Recall= 90.4 %
# precision= 98.4 %
# F score= 94.1 %


#write(capture.output(confusionMatrix(ConfMatrix_bayes_mat)
),"bayes confusion_matrix R.txt")
```

```
# naive bayes
library(e1071)
nb_model=naiveBayes(target~.,data=train1)
#(summary(nb_model))
nb_predicion=predict(nb_model,newdata=test2[,-1],type="clas
s")
# prediction of test cases
nb_test_pred=predict(nb_model,newdata=test,type="class")
# model evaluation
ConfMatrix_bayes_mat = table(test2$target,nb_predicion)
confusionMatrix(ConfMatrix_bayes_mat)
# Accuracy= 92.1 %
# FNR= 6.8 %
# Recall= 93.1 %
# precision=  98.4 %
# F score= 95.6 %
```

```
# random forest classifier
library(randomForest)
rf_model=randomForest(target~.,train1,importance=TRUE,ntree
=50)
#Extract rules from random forest
#transform rf /object to an inTrees' format
library(inTrees)
treeList = RF2List(rf_model)
#Extract rules
```

```
rules= extractRules(treeList, train1[,-1])
#Visualize some rules
rules[1:2,]
#Make rules more readable:
readrules = presentRules(rules, colnames(train1))
readrules[1:2,]
#Predict test data using random forest model
rf_Predictions = predict(rf_model, test2[,-1])
# prediction of test cases.
rf_Pred_test = predict(rf_model, test)
# model evaluation
ConfMatrix_random_mat = table(test2$target,rf_Predictions )
confusionMatrix(ConfMatrix_random_mat)
#write(capture.output(rf_model),"rf_model R.txt")

# Accuracy=89.76 %
# FNR= 10.2 %
# Recall= 89.7 %
# Precision= 99.9 %
# F score= 94.5 %

# note:-
# train1 is the training dataset.
# test2 is the testing dataset.
# train1 and test2 are created for the sake of calculating
confusion matrix.
# test is the actual dataset which must be predicted.
```

Complete python file

This below code file comprised of prediction results of test data set by training the test data set.

```
In [1]:# importing all the necessary data operation
libraries.
import os
os.chdir("D:/R and PYTHON files/data set/project 2")
os.getcwd()
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
In [2]:
# loading the train data
train_data=pd.read_csv("train.csv",sep=',')
In [3]:
# loading the test data
test_data=pd.read_csv("test.csv",sep=',')
In [4]:
# exploratory data analysis
train_data.shape
Out[4]:
(200000, 202)
In [5]:
test_data.shape
Out[5]:
(200000, 201)
In [50]:
# train_data.head()
```

```
train_data.describe()
. . .
In [51]:
# test_data.head()
test_data.describe()
. . .
In [8]:
train_data['target'].value_counts().plot.bar();
. . .
%matplotlib inline plt.boxplot(test_data['var_190'])
In [9]:
train_data['target'].value_counts()# this seems to be a
class imbalance data set.
Out[9]:
0    179902
1     20098
Name: target, dtype: int64
In [49]:
sns.violinplot(data=train_data,x="target",y="var_1").get_fi
gure().savefig("violin1.png")
. . .
In [11]:
sns.violinplot(data=train_data,x="target",y="var_110").get_
figure().savefig("violin100.png")
. . .
In [4]:
numerical_cols=["ID_code","target"]
numeric_features=train_data.drop(numerical_cols,axis=1)
#y=train_data["target"]
numeric_features.head()

. . .
```

```
In [5]:
df_corr=numeric_features
f,ax=plt.subplots(figsize=(7,5))
corr=df_corr.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=True),
           square=True,
ax=ax).get_figure().savefig('heatmap.png')
. . .
In [6]:
train_data.isnull().sum().sum(),test_data.isnull().sum().su
m()
numeric_features.shape
Out[6]:
(200000, 200)
In [44]:
# saving the fig in directory

#plt.hist(train_data['var_0'])
#plt.savefig('hist_v0.png')

plt.hist(train_data['var_100'])
plt.savefig('hist_v100.png')

#plt.hist(train_data['var_10'])
#plt.savefig('hist_v10.png')

#plt.hist(train_data['var_120'])
#plt.savefig('hist_v120.png')
. . .
In [20]:
# checking the distribution of the features.
```

```
for i,col in enumerate (numeric_features):
   plt.figure(figsize=(10,30))
   plt.subplot(50,4,i+1)
   plt.hist(train_data[col])
   plt.title(col)


. . .
In [26]:
# checking the distribution of target variable for each
feature.
for i,col in enumerate (numeric_features):
   plt.figure(figsize=(20,40))
   plt.subplot(50,4,i+1)

plt.hist(train_data[train_data["target"]==0][col],alpha=0.5
,label='0',color='green')

plt.hist(train_data[train_data["target"]==1][col],alpha=0.5
,label='1',color='yellow')
   plt.title(col)
. . .
In [20]:
# mean frequency
plt.figure(figsize=(20,8))
numeric_features.mean().plot('hist').get_figure().savefig('
mean_freq.png')
plt.title('mean frequency');


. . .
In [45]:
#median frequency
plt.figure(figsize=(20,8))
```

```
numeric_features.median().plot('hist').get_figure().savefig
('mean_freq.png')
plt.title('median frequency');

. . .
In [46]:
# standard deviation freq.
plt.figure(figsize=(20,8))
numeric_features.std().plot('hist').get_figure().savefig('m
ean_freq.png')
plt.title('standard deviation frequency');

. . .
In [47]:
# checking skewness
plt.figure(figsize=(20,8))
numeric_features.skew().plot('hist').get_figure().savefig('
mean_freq.png')
plt.title('skew frequency');

. . .
In [48]:
# kurtosis
plt.figure(figsize=(20,8))
numeric_features.kurt().plot('hist').get_figure().savefig('
mean_freq.png')
plt.title('kurtosis frequency');

. . .
In [5]:
# preparing the data for logistic regression model model
columns=["ID_code","target"]
```

```
x=train_data.drop(columns,axis=1)
In [6]:
# importing all the necessary library for model
import sklearn
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
import statsmodels.api as sm

In [52]:
# logistic regression
logistic_model = sm.Logit(train_data['target'],x).fit()
logistic_model.summary()
. . .
In [7]:
col=["ID_code"]
xtest=test_data.drop(col,axis=1)

In [53]:
prediction_logistic=logistic_model.predict(xtest)
In [75]:
prediction_logistic.describe()


. . .
In [8]:
# preparing the data for decision tree and randon forest
y=train_data["target"]
In [20]:
# decision tree classifier
train_x, val_x, train_y, val_y = train_test_split(x, y,
random_state=1)
tree_model =
tree.DecisionTreeClassifier(criterion='entropy').fit(train_
x,train_y)
In [108]:
print(tree_model)
. . .
In [21]:
predictions_tree=tree_model.predict(xtest)
predictions_tree
Out[21]:
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
In [14]:
# random forest classifier
train_x, val_x, train_y, val_y = train_test_split(x, y,
random_state=1)
rfc_model
=RandomForestClassifier(random_state=0).fit(train_x,
train_y)
C:\Users\ELCOT\Anaconda3\lib\site-packages\sklearn\ensemble
\forest.py:245: FutureWarning: The default value of
n_estimators will change from 10 in version 0.20 to 100 in
0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

In [15]:
predict_rfc=rfc_model.predict(xtest)
predict_rfc
Out[15]:
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
In [16]:
# variable importance
import eli5
from eli5.sklearn import PermutationImportance
```

```
perm = PermutationImportance(rfc_model,
random_state=1).fit(val_x, val_y)


In [60]:
eli5.show_weights(perm, feature_names =
val_x.columns.tolist(), top=200)
. . .
In [27]:
# What can be inferred from the above?
# As you move down the graph, the importance of the feature
decreases.
# The features that are shown in green indicate that they
have a positive impact on our prediction
# The features that are shown in white indicate that they
have no effect on our prediction
# The features shown in red indicate that they have a
negative impact on our prediction
# The most important feature was Var_110.
```

The below code file is created in python to build the confusion matrix.

Note:- Test data set is divided into two parts, the training and testing data set for the building of confusion matrix and selecting the best model.

```
# importing all the necessary data operation libraries.
import os
os.chdir("D:/R and PYTHON files/data set/project 2")
os.getcwd()
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
# to display all the columns of the dataframe in the
notebook
pd.pandas.set_option('display.max_columns', None)
from sklearn import tree
# loading the train data
data=pd.read_csv("train.csv",sep=',')
data.head()
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split
y=data['target']
x=data.drop(["ID_code","target"],axis=1)
uracy.
# splitting the train data set for model building and
finding accuracy.
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.
1)
sel_ = SelectFromModel(Lasso(alpha=0.005, random_state=0))
# remember to set the seed, the random state in this
function
sel_.fit(xtrain, ytrain)

sel_.get_support()
# let's print the number of total and selected features

# this is how we can make a list of the selected features
selected_feat = xtrain.columns[(sel_.get_support())]
```

```
# let's print some stats
print('total features: {}'.format((xtrain.shape[1])))
print('selected features: {}'.format(len(selected_feat)))
print('features with coefficients shrank to zero:
{}'.format(
    np.sum(sel_.estimator_.coef_ == 0)))
total features: 200
selected features: 160
features with coefficients shrank to zero: 40
# print the selected features
selected_feat
# this is an alternative way of identifying the selected
features
# based on the non-zero regularisation coefficients:
selected_feats = xtrain.columns[(sel_.estimator_.coef_ !=
0).ravel().tolist()]
selected_feats
# now we save the selected list of features
pd.Series(selected_feats).to_csv('selected_features.csv',
index=False)
features = pd.read_csv('selected_features.csv',
header=None)
features = [x for x in features[0]]
features = features

features
# reduce the train and test set to the desired features

xtrain = xtrain[features]
xtest = xtest[features]
xtrain.shape,xtest.shape
# train the model
```

```
lin_model = Lasso(alpha=0.005, random_state=0) # remember
to set the random_state / seed
lin_model.fit(xtrain, ytrain)
pred_model=lin_model.predict(xtest)
tree_model =
tree.DecisionTreeClassifier(criterion='entropy').fit(xtrain
,ytrain)
predictions_tree=tree_model.predict(xtest)
from sklearn.metrics import confusion_matrix
CM = confusion_matrix(ytest,predictions_tree)
CM=pd.crosstab(ytest,predictions_tree)
CM
#let us save TP, TN, FP, FN
TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]

from sklearn.metrics import accuracy_score
accuracy_score(ytest,predictions_tree)*100
83.565
from sklearn.ensemble import RandomForestClassifier
rfc_model
=RandomForestClassifier(random_state=0).fit(xtrain,ytrain)
rfc_model
rfc_pred=rfc_model.predict(xtest)
rfc_pred
mc = confusion_matrix(ytest,rfc_pred)
mc=pd.crosstab(ytest,rfc_pred)
mc

#let us save TP, TN, FP, FN
```

```
tn = mc.iloc[0,0]
fn = mc.iloc[1,0]
tp = mc.iloc[1,1]
fp = mc.iloc[0,1]
accuracy_score(ytest,rfc_pred)*100
((tp+tn)*100)/(tp+tn+fp+fn)
((TP+TN)*100)/(TP+TN+FP+FN)
```