Project-1 : Santander Customer Transaction Prediction

INTRODUCTION

Santander Bank poses a challenge in order to help them with the problem of identification of the customers who will make a transaction with the bank in future, irrespective of the amount of money transacted previously with the bank.

The data that is provided for solving the problem is masked completely with only numeric values.

- The data is anonymous with no Customer details been revealed to the participants of the competition.
- The data sheet contains 200000 rows for both train and test data.
- The Train Data set has 202 columns with 200 columns having values for var_1 to var_200, one column for ID code and one column for target, which are the outcome of the transaction.
- The same columns are present for test data except for the target column, which we needs to predict as per the problem statement.

A model is designed that predicts the target customers as per the ID_code which will be going to do transaction.The project is based on the Binary Classification type of Data Science problem.

DATA SET DESCRIPTION

In the problem we are provided with an anonymized dataset containing numeric feature variables (199 independent variables of int datatype), binary target column (which is dependent variable having binary values 0/1) and the string ID column for each data point for the Train set.

For the Test set, we are provided with numeric feature variables (199 independent variables) and the string ID column. The test dataset does not have the target column as this column we have to predict using the Train Data set.

The data files are provided in the form of CSV format these CSV format files are used in both R and python.

Software Requirement

A). R 3.3.3 for 64 bit

B). Anaconda 3.4.4.0 for 64 bit

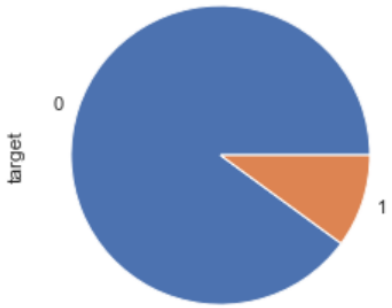
EXPLORATORY DATA ANALYSIS

All the required libraries are installed and loaded in both the environment.The working directory is set. Train and test dataset files are loaded in both the environment.ID_code and target is factor value in R and int64 in python. Remaining all the independent variables (var_01 to var_199) are float/numerical.

Step1: Counted observations as per the target class.

```
In [10]: #plotting pie chart for target class
         train['target'].value_counts().plot(kind='pie', figsize=(4,4))

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1fd0e736b88>
```



Observation:

Around 90% of our data has 0 (customer did not do a transaction), and around 10% of the data (customer did do the transaction). This shows that the problem in hand is a binary classification problem and this maks the data is very imbalanced.

Exact count of the target variables are as follows:

Counting observations as per the target class

```
In [9]: train.target.value_counts()

Out[9]: 0    179902
        1     20098
        Name: target, dtype: int64
```

Step 2: Dealing with missing values

```
In [90]: train.isnull().sum()

Out[90]: ID_code    0
         target     0
         var_0      0
         var_1      0
         var_2      0
                   ..
         var_195    0
         var_196    0
         var_197    0
         var_198    0
         var_199    0
         Length: 202, dtype: int64
```

```
In [91]: test.isnull().sum()

Out[91]: ID_code    0
         var_0      0
         var_1      0
         var_2      0
         var_3      0
                   ..
         var_195    0
         var_196    0
         var_197    0
         var_198    0
         var_199    0
         Length: 201, dtype: int64
```

After doing Missing Value Analysis it is observed that there are no missing values present in both train and test data.

Step3*: Correlation Analysis*

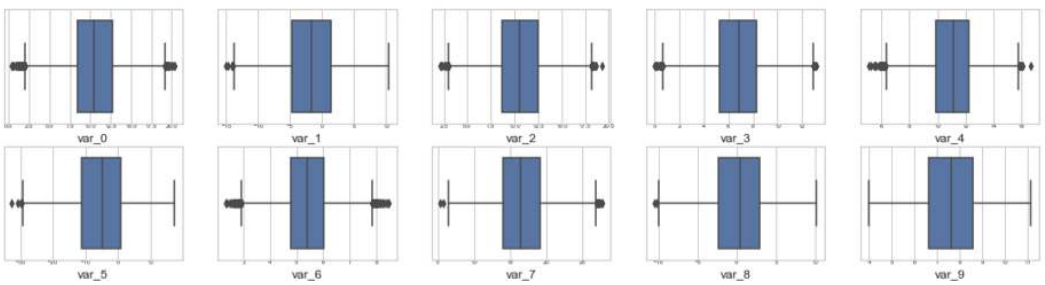
Correlation analysis is a statistical method used to evaluate the strength of relationship between two quantitative variables. A high **correlation** means that two or more variables have a strong relationship with each other, while a weak **correlation** means that the variables are hardly related.If value of correlation coefficient is +1 or -1, then variables are strongly connected to each other and when correlation is 0, then variables are not correlated to each other and are independent. As per the observation by plotting heat map it can be observed that the variables are not correlated with each other and dependent variable.

Step4: *Outlier Analysis*

An **outlier** is a data point that differs significantly from other observations. It may be due to variability in the measurement or may indicate experimental error; the latter are sometimes excluded from the data set.It can cause serious problems in **statistical** analyses. It can also be known as Noise in dataset so needs to detect the outliers. **Outliers** which are present in the variables are detected by box plot analysis.

```
In [16]:  # From var_0 to var_49 (training set)
features = train.columns.values[2:52]
plot_boxplot(train,features)
```

```
<Figure size 432x288 with 0 Axes>
```



Box plots are a type of graph that can help visually organize data. To graph a **box plot** the following data points must be calculated: the minimum value, the first quartile, the median, the third quartile, and the maximum value

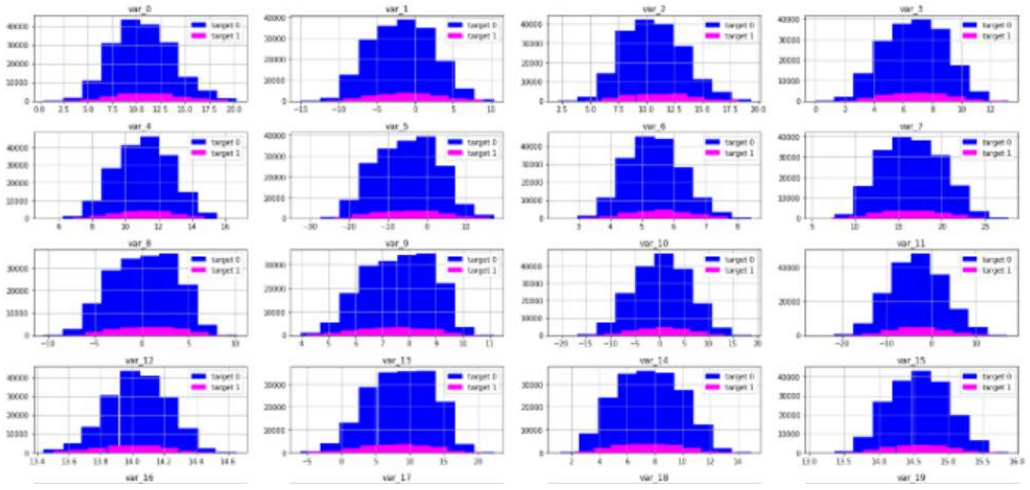
Observed that there are certain outliers in both train and test dataset. In order to treat outliers we can directly remove them or can impute them as well, as directly removing outliers may leads to the loss of information. To decide that either we should remove the outliers or to impute them we needs to do correlation analysis as if there will be any correlation present between the dataset we needs to impute them otherwise not. As there is no correlation between the variables we can simply drop the outliers.

After dropping the outliers the shape of dataset:

```
In [101]: #shape of train and test data after removal of outliers
          train.shape,test.shape

Out[101]: ((175073, 202), (174011, 201))
```

Step 5: Once we done with missing value, outlier analysis and correlation analysis , we try to find out the *distribution across the variables.*

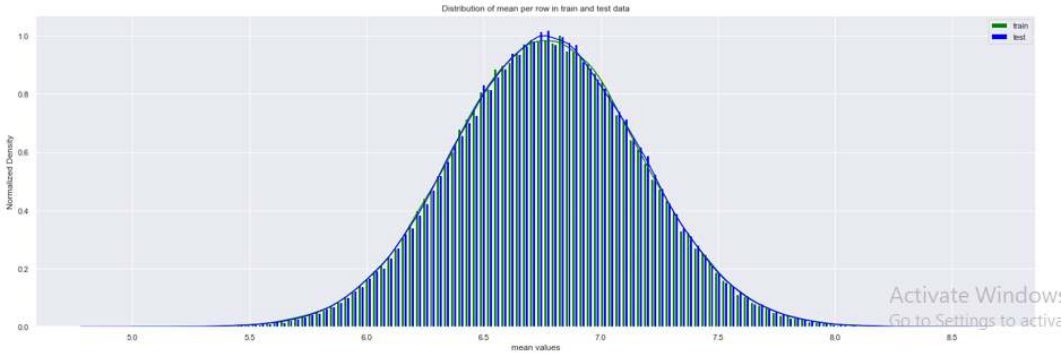


We can observe from this that there is normal distribution for all the variables for both the outcome 1 and for outcome 0.

Step6: *Descriptive Statistics of train and test dataset*

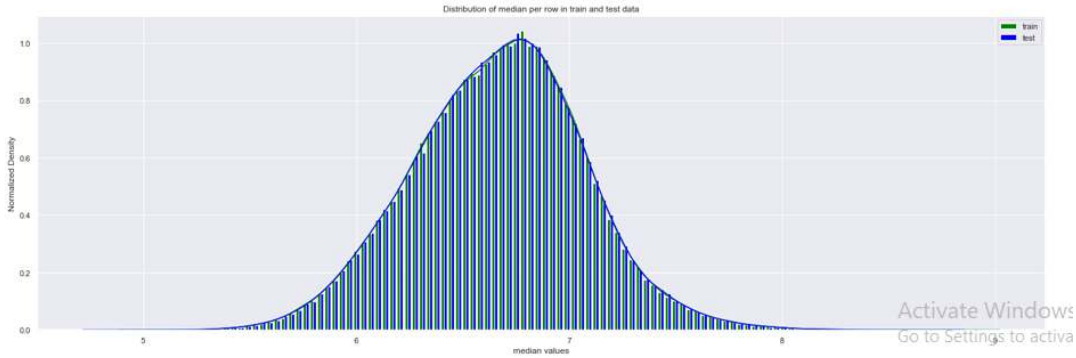
1. Mean

```
In [30]: plot_stats(train_mean_rowwise,test_mean_rowwise,'mean','row')
```



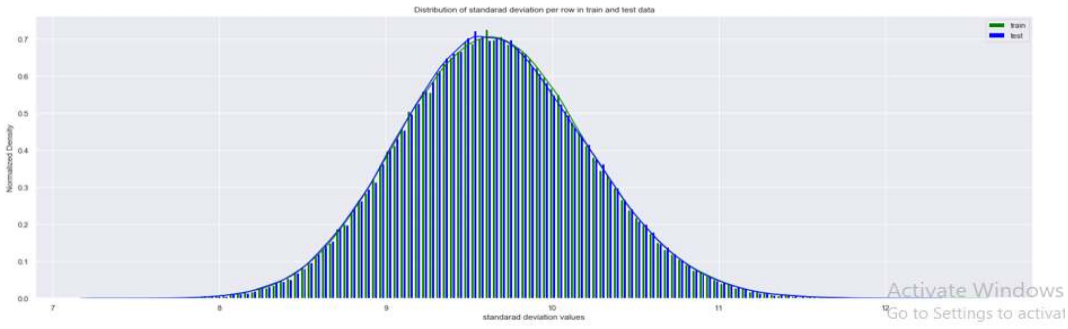
2. Median

```
In [31]: plot_stats(train_median_rowwise,test_median_rowwise,'median','row')
```



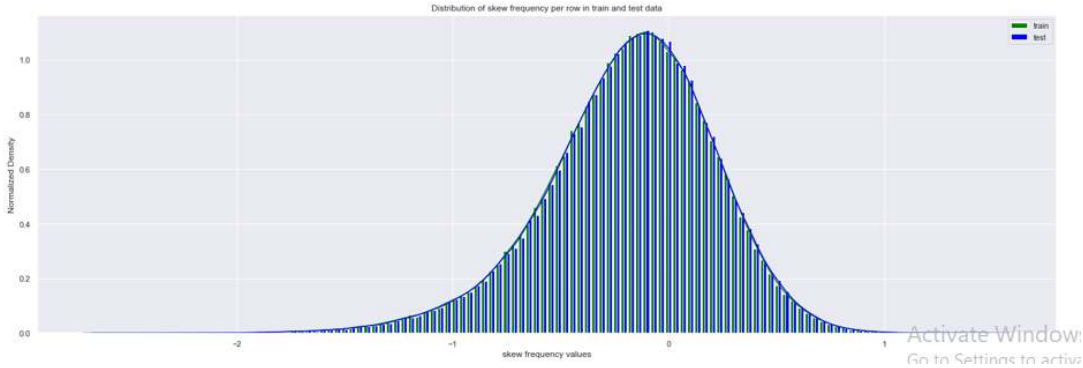
3. Standard Deviation

```
In [32]: plot_stats(train_std_rowwise,test_std_rowwise,'standarad deviation','row')
```



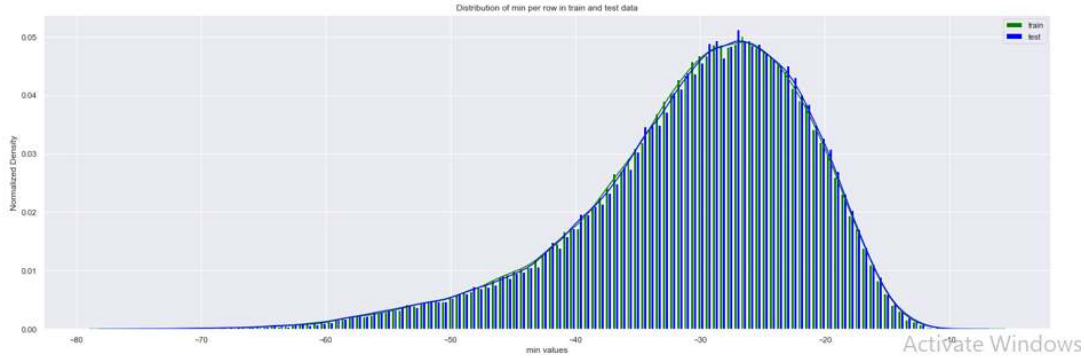
4. Skewness

```
In [33]: plot_stats(train_skew_rowwise,test_skew_rowwise,'skew frequency','row')
```



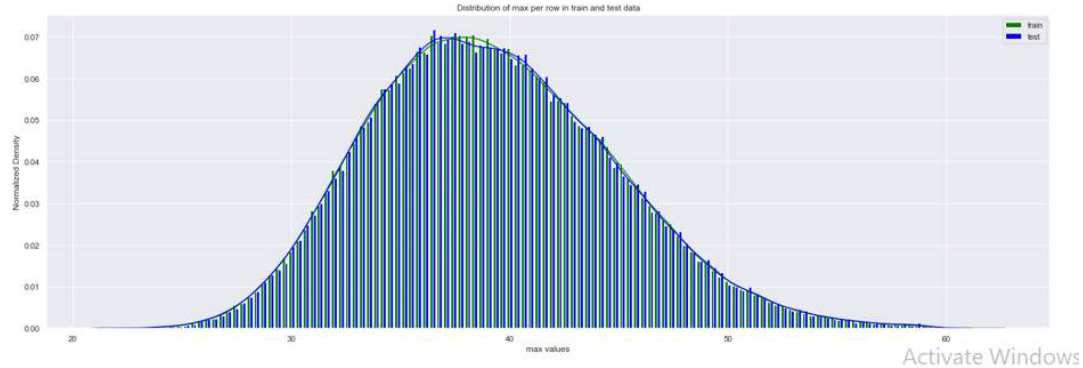
5. Min.

```
In [34]: plot_stats(train_min_rowwise,test_min_rowwise,'min','row')
```



6. Max.

```
In [35]: plot_stats(train_max_rowwise,test_max_rowwise,'max','row')
```



We can make few observations here:

1. Standard deviation is relatively large for both train and test variable data.

2. min, max, mean, median, sdt values for train and test data looks quite close.

3. Mean values are distributed over a large range.

4. Moreover mean and median have simmilar distribution.

5. Both train and test are negatively skewed.

6. We have 200 variables in both train and test set which don't have any correlation between them.

7. All the histograms of 200 numerical features resemble the shape of normal distribution

FEATURE SCALING

The process of transforming raw data into a standard scale and remove variance from the data.

$$\text{Standardized variable} = (x-\mu)/\text{Std. Dev.}$$

DIMENTIONALITY REDUCTION TECHNIQUE

We have performing outlier analysis and correlation analysis let's apply dimension reduction technique so that less informative features can be removed.

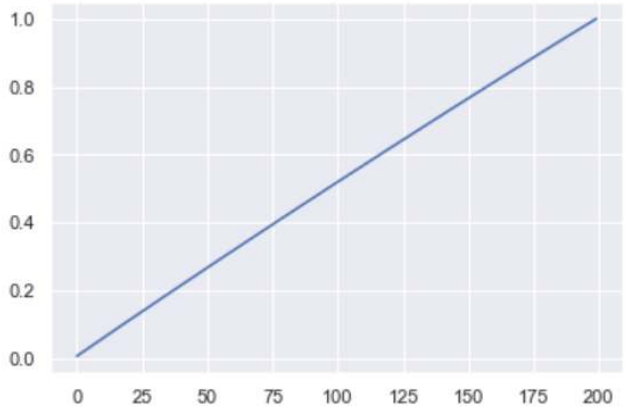
Principal Component Analysis(PCA)

It is a dimension reduction technique that reduces **less-informative 'noise'** features and create new features that retains the original variance of data points and are very informative. These features are less in number as compared to original features.As PCA is sensitive to variance and different scales, so standardizing will help PCA perform better.

However, since we found that the correlation between different features in the training dataset is not that significant, so using PCA might not be meaningful.

```
plt.plot(np.arange(200), np.cumsum(explained_variance))
```

```
Out[39]: [<matplotlib.lines.Line2D at 0x1fd32ffe188>]
```



Observation:-

The line of cumulative sums of explained variance ratio when we PCA the data set, it indicates that the dataset has already undergone PCA (get straight line ie y=x).

We have to go with all 200 variables

FEATURE ENGINEERING

In feature engineering, we create new features that make machine learning algorithms work by using domain knowledge as features are important for predictive models and influence results.

Here, the independent variable's names are not significant so we can't we able to apply domain knowlwdge. However we can create new features as per our descriptive statistics observations.

New features can be added to our dataset as (Sum,Mean, Median, Standard Deviation,Skewness,Min. And Max. Values)

MODEL DEVELOPMENT

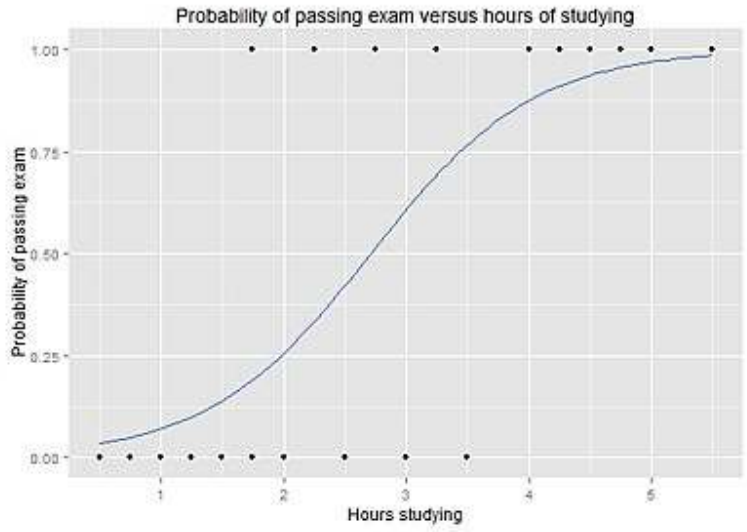
1. ***Logistic Regression***:

Logistic regression is a technique borrowed by machine learning from the field of statistics, a go-to method for binary classification problems (problems with two class values i.e. 0 and 1).

The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

$$1/(1+e^{-value})$$

Where '**e**' is the base of the natural logarithms (Euler's number or the EXP() function in) and value is the actual numerical value that we want to transform.



Representation of user Logistic Regression:

Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value. Below is an example logistic regression equation:

$$Y=e^{(bo+b1x)}/(1+e^{(b0+b1x)})$$

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

In order to apply Logistic Regression we needs to split our Train data into Training set and Test set. We train our model on the training set and test the accuracy of the model on test dataset. After applying Logistic Regression Model on our training dataset the confusion matrix and Accuracy of train data and test data is plotted as per the AUC, ROC curve.

Confusion Matrix: Confusion Matrix shows how confused our model is.

```
In [120]: cm
```

Out[120]:

col_0	0	1
target		
0	46770	621
1	3765	1366

Here we can observe that the confusion matrix after applying the logistic Regression is:

True Positive:46770 (means the model has predicted 0 and actual value was 0 itself)

True Negative:1366 (means the model has predicted 1 but the true value was 1)

False Positive: 621 (means the model has predicted 1 while the true value was 0)

False Negative:3765 (means the model has predicted 0 while the true value was 1)

Receiver Operating Characteristics:

A **receiver operating characteristic curve**, or **ROC curve**, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

The true-positive rate is also known as sensitivity, recall or *probability of detection* in machine learning.

The false-positive rate is also known as *probability of false alarm* and can be calculated as (1 − specificity).

It can also be thought of as a plot of the power as a function of the Type I Error of the decision rule (when the performance is calculated from just a sample of the population, it can be thought of as estimators of these quantities).

The ROC curve is thus the sensitivity as a function of fall-out.

In general, if the probability distributions for both detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function (area under the probability distribution from $-\infty$ to the discrimination threshold) of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability on the x-axis.

1. Sensitivity, recall, true positive rate

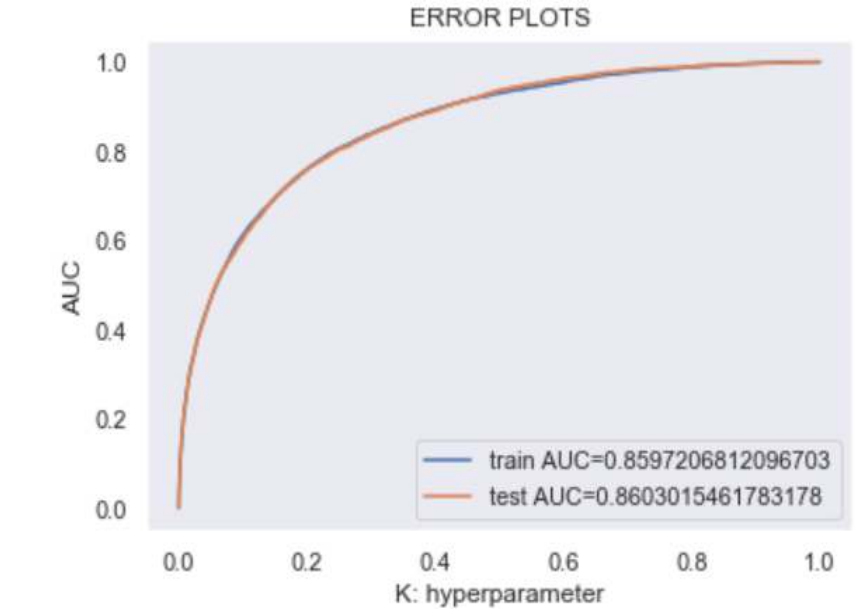
   TPR= TP/P = TP/TP+FN

2. Specificity, selectivity, true negative rate

   TNR= TN/N = TN/TN+FP

3. Precision or positive predictive value

   PPV= TP/TP+FP



Observation Of Logistic Regression Model:

Train AUC = 0.859

Test AUC=0.860

2. Gaussian Naïve Bayes:

Naive Bayes can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution. This extension of naive Bayes is called Gaussian Naive Bayes. Other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because you only need to estimate the mean and the standard deviation from your training data.

Representation of Gaussian Naïve Bayes:

Calculated the probabilities for input values for each class using a frequency. With real-valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution. This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

Learning a Gaussian Naïve Bayes Model from Data:

Bayes Theorem: $P(A|B) = (P(B|A)*P(B))/P(A)$

This is as simple as calculating the mean and standard deviation values of each input variable (x) for each class value.

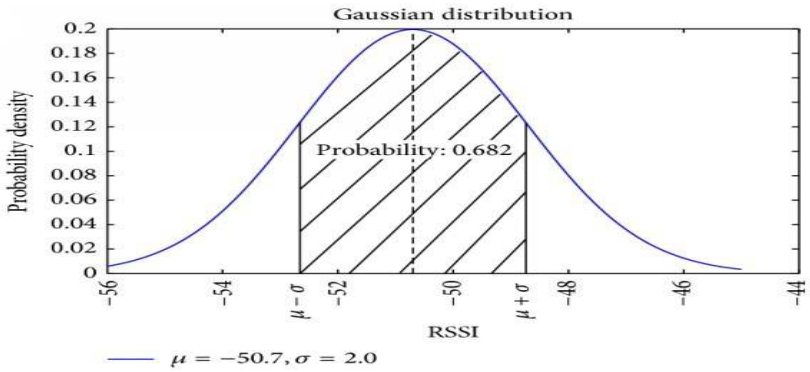
$Mean(x)=1/n*sum(x)$

Where **n** is the number of instances and **x** are the values for an input variable in your training data. We can calculate the standard deviation using the following equation:

$Standard\ deviation(x)=\sqrt{1/n*sum(xi-mean(x^2))}$

Making Predictions from Gaussian Naïve Bayes:

Probabilities of new x values are calculated using the Gaussian Probability Density Function.



When making predictions these parameters can be plugged into the Gaussian PDF with a new input for the variable, and in return the Gaussian PDF will provide an estimate of the probability of that new input value for that class.

PDF(x,mean,SD)=1/√(2*PI)*SD) * exp(-(x-mean^2/2-SD^2))

Where pdf(x) is the Gaussian PDF, sqrt() is the square root, mean and sd are the mean and standard deviation calculated above, PI is the numerical constant, exp() is the numerical constant e or Euler's number raised to power and x is the input value for the input variable.

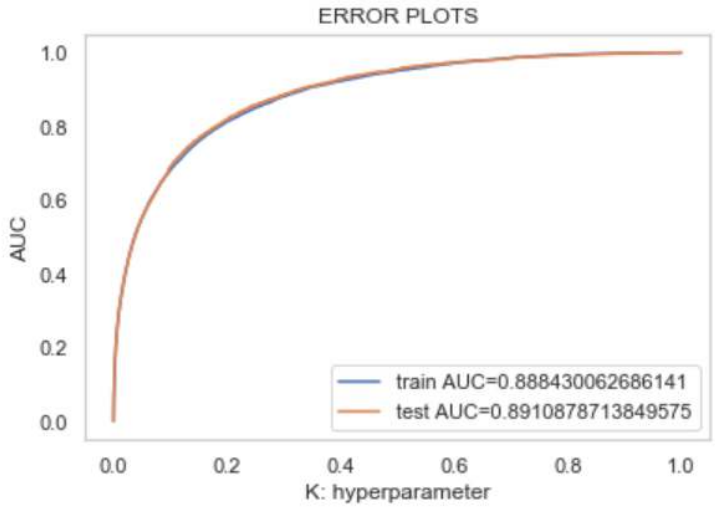
We can then plug in the probabilities into the equation above to make predictions with real-valued inputs.

Confusion Matrix:

Out[140]:

col_0	0	1
target		
0	46770	621
1	3765	1366

AUC/ROC curve:



Hence we can observe that after applying Navie Bayes ML algorithm:

Train AUC = 0.888

Test AUC = 0.891

- **Ensemble models**

  Ensemble models are nothing but aggregation of a number of Weak Learners.

  There are 2 types of ensemble learning algorithms

  1. Bagging Algorithms: Bagging involves having each model in the ensemble vote with equal weight for the final output. In order to promote model variance, bagging trains each

model in the ensemble using a randomly drawn subset of the training set

2. Boosting Algorithms: Boosting involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models mis-classified.

3. Random Forest:

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds.

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. **The reason for this wonderful effect is that the trees protect each other from their individual errors** (as long as they don't constantly all occur in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

So the prerequisites for random forest to perform well are:

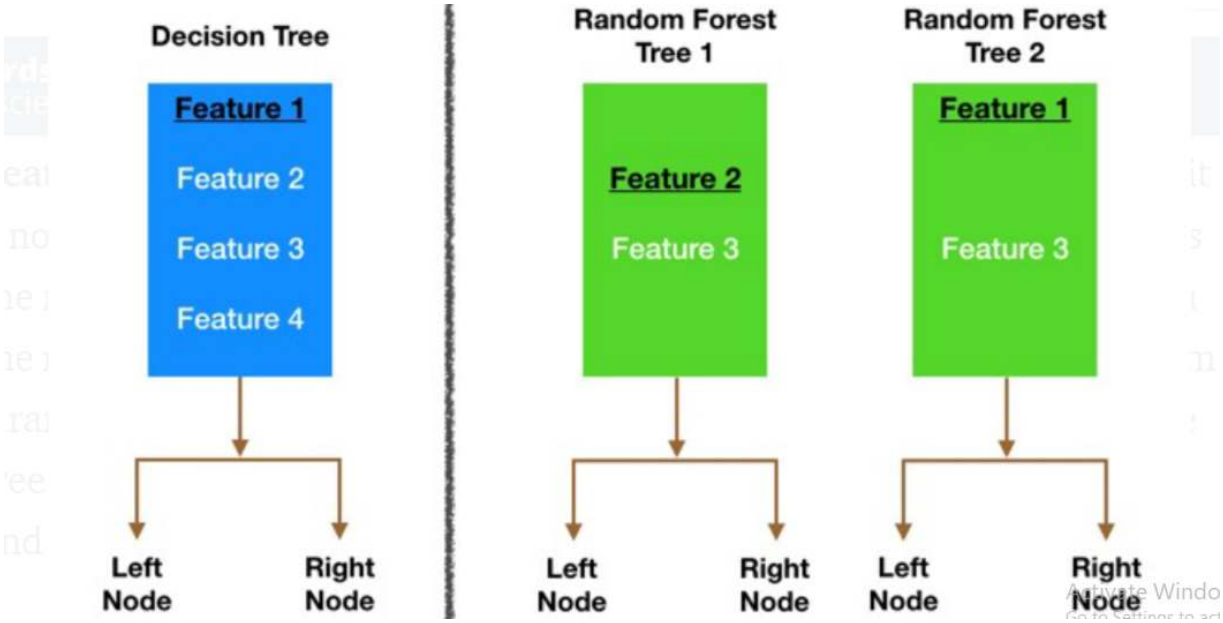
1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.

2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

3. Ensuring that the Models Diversify Each Other

4. So how does random forest ensure that the behavior of each individual tree is not too correlated with the behavior of any of the other trees in the model? It uses the following two methods:

5. **Bagging (Bootstrap Aggregation) — Decisions trees are very sensitive to the data they are trained on — small changes to the training set can result in significantly different tree structures.** Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This process is known as bagging.



Feature Randomness — In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the

observations in the left node vs. those in the right node. In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification.

So in our random forest, we end up with trees that are not only trained on different sets of data (thanks to bagging) but also use different features to make decisions.

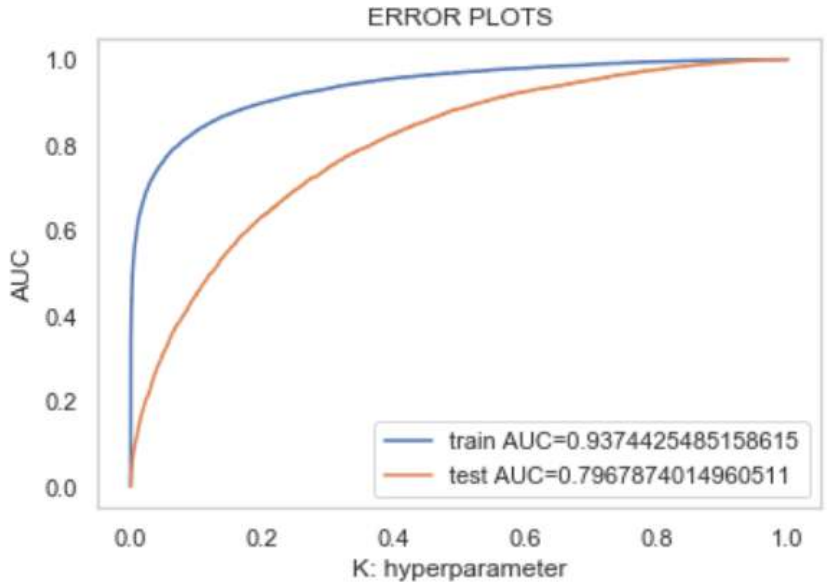
The random forest is a classification algorithm consisting of many decisions trees. ***It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees*** *whose prediction by committee is more accurate than that of any individual tree.*

Confusion Matrix:

Out[166]:

col_0	0	1
target		
0	47368	23
1	5107	24

AUC/ROC Curve :



Hence, After applying Random Forest we can observe that:

Train AUC = 0.9374

Test AUC = 0.7967

4. **Light GBM:**

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.Since, it is based on decision tree algorithms, it splits the tree leaf wise with the best fit. With large datasets, light gbm performs very well.

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

· Faster training speed and higher efficiency.

· Lower memory usage.

· Better accuracy.

· Support of parallel and GPU learning.

· Capable of handling large-scale data.

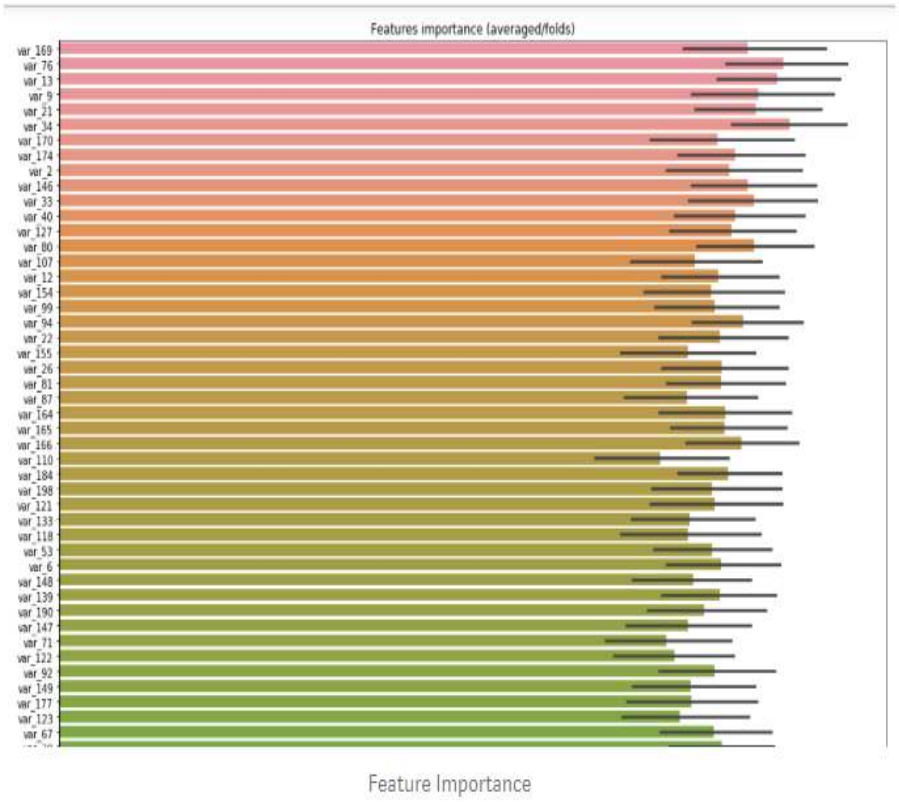
Difference from GBDT and Decision Trees:

Light GBM grows tree vertically while other algorithm grows trees horizontally which means that Light GBM grows tree **leaf-wise** while other algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

Hyperparameters of our Light GBM Model:

```
param = {
    'bagging_freq': 5,
    'bagging_fraction': 0.4,
    'boost_from_average':'false',
    'boost': 'gbdt',
    'feature_fraction': 0.05,
    'learning_rate': 0.01,
    'max_depth': -1,
    'metric':'auc',
    'min_data_in_leaf': 80,
    'min_sum_hessian_in_leaf': 10.0,
    'num_leaves': 13,
    'num_threads': 8,
    'tree_learner': 'serial',
    'objective': 'binary',
    'verbosity': 1
}
```


Feature Importance:



COMPARISION BETWEEN ALL THE MODELS

1. Logistic regression — Test AUC: 0.8635
2. Gausian Naive Bayes — Test AUC : 0.8930
3. Random Forest classifier — Test AUC: 0.796
4. Light GBM — Test AUC : 0.8978

After compairing all the models we can find that Light GBM performs well among all.