Santander Customer Transaction Prediction

SUMITTED BY:
AYUSH KUMAR JAIN

CONTENT

1. PROBLEM STATEMENT

At Santander, mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals. Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

Problem Statement -

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

Data Set contains

1) test.csv
2) train.csv
Attributes in Train data:

The Train data contains total 200 independent numerical variable that are var_0, var_1…… var_199. Other than these variables it has Column with customer ID named as ID_column and a target variable with class 0 and 1.

Total number of observations provided in train data are 200000.

Attributes in Test data:

The Test data contains total 200 numerical variable same as provided in Train data, it also has customer ID column as ID_code. We have to predict the class of target variable for this data set.

Hence, our target variable is categorical variable have class as 0,1. We can see that this is a Classification problem.

2. Data Pre-processing:

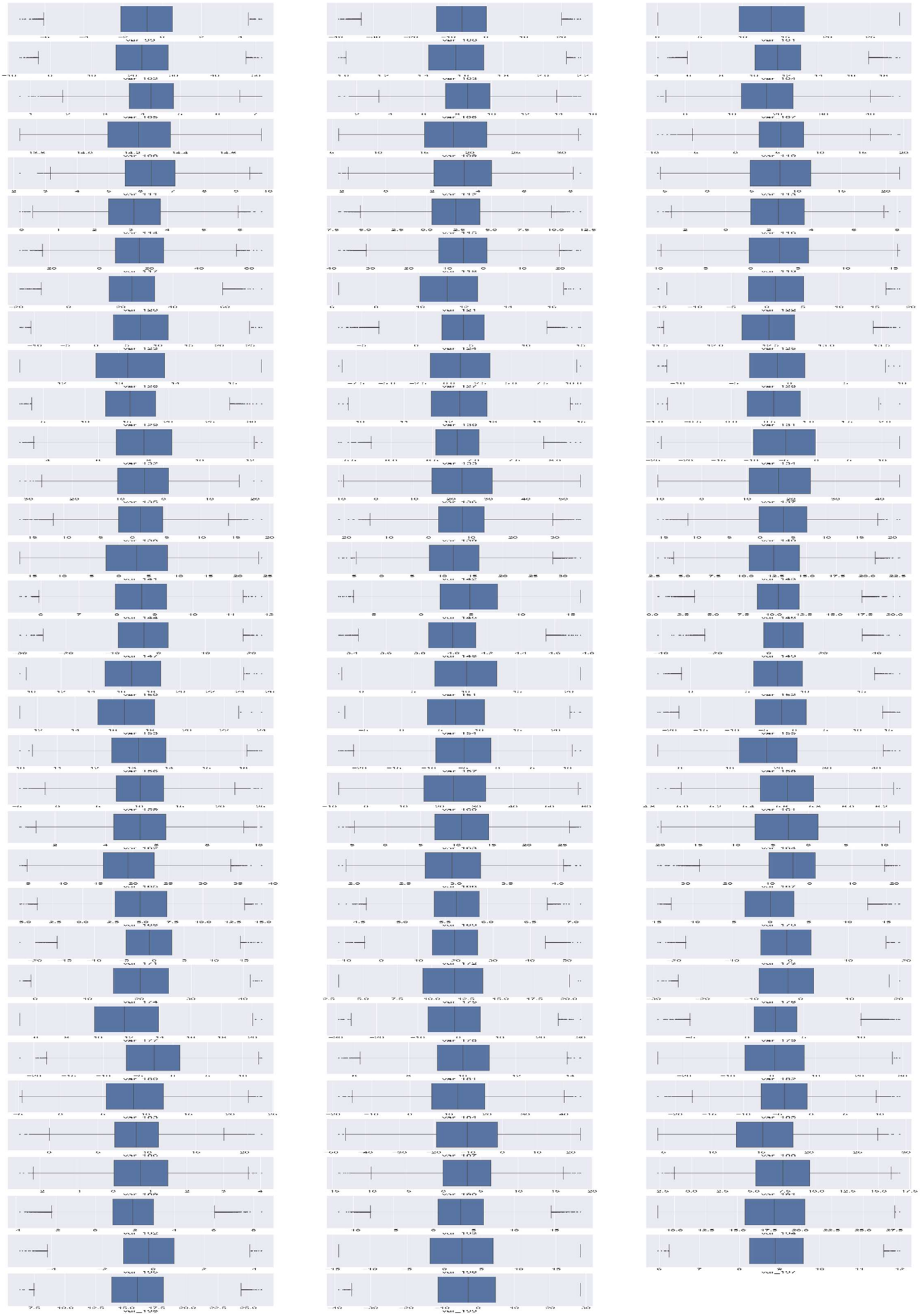
In this process we try to understand data more clearly to find out about missing values in data, outliers in data. After that we try to understand the behaviour (like how they vary, there mean, standard deviation) of values of numerical variables in train and test data. As we have highly dimensionality in our data so we also try some technique like principal component analysis and random forest to reduce this huge data.

2.1 Missing value analysis:

On checking for missing values in train and test data we found out there is no missing values in these data.

2.2 Outlier Analysis:

We are doing Outlier analysis on train data. We have plot boxplot of all numerical variable as shown:

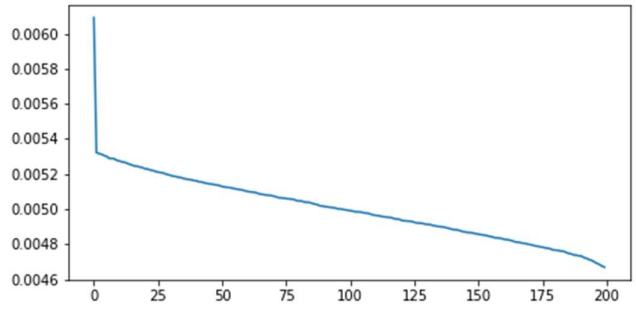


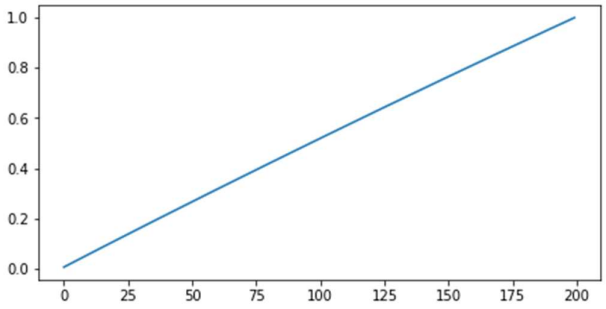
As we can see some of the variable does not have outliers but most of the variables have outlier. What we can do is we can replace them with NA than impute them using different missing value imputation method or we can drop them. On analysis or building models we find out we are getting better results when we build model on data that we get after replacing the missing values with NA then imputing them using mean of the variable.

2.3 Dimensionality Reduction:

2.3.1 Correlation analysis:
We calculate correlation between each and every independent numerical variable, from results we can see that no independent variable is highly correlated to other.
`Max.cor.:0.0676, Min. cor.: -0.0801`

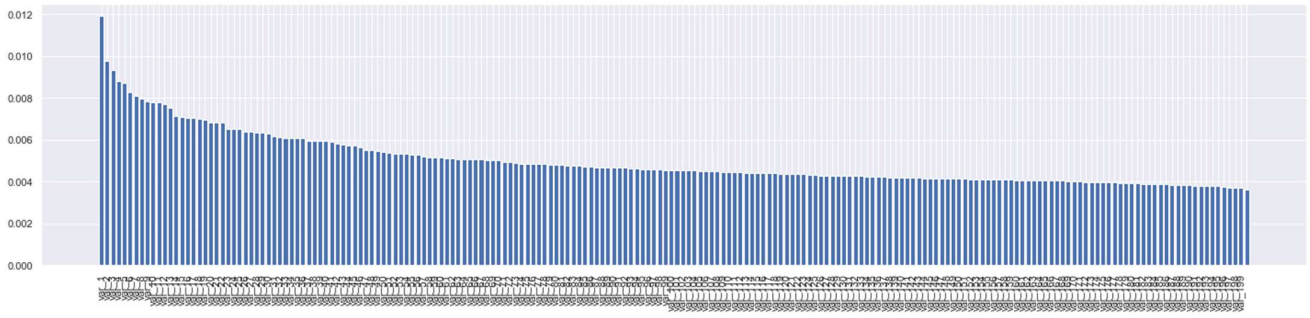
2.3.2 Principal component analysis:
On doing principal component analysis we find out curve for cumulative variance curve is straight line that mean that data is already gone through PCA.





2.3.3 Random Forest:

We made a model for predicting target variable using all independent variable than we plot feature importance curve as shown below:



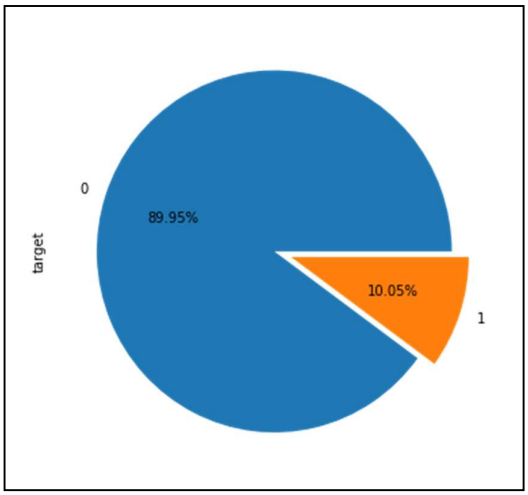
We can see that no variable has such low importance. We have to go with all 200 numeric independent variables.

2.4 Data Visualization:

A) Target Variable:

We plot pie chart of target variable as shown below, target variable has imbalancing in class.

Way to deal with Imbalanced data:



1) **Random Under sampling**: Randomly dropping majority class cases from main data. This may cause loss of important information.
2) **Random Over sampling**: Randomly replicating minority class cases in main data. This may cause overfitting. Still we try to **build model on over sampled data in R**.
3) **SMOTE**: Exact replica of minority class cases in main data. This method does require high memory in system when we are working with high dimensional data, as we have high dimension in data, we do not use this method.
4) **Ensemble Techniques**: using Bagging and boos ting bases techniques (Ada and gradient boost techniques). **We use XG boost in both R and python**.

2.5 Observations from visualization: (Not adding visualization due to memory shortage, for visualization please see python code file)

1) We plot distribution of all numeric independent data of train and test; from this we find out that almost all independent variable has normally distributed in both train and test.
2) standard deviation is relatively large for both train and test variable data.
3) mean, std, min, max values for train and test data looks quite closely.
4) Mean values are distributed over a large range.

2.6 Feature scaling:

As we find out that numerical independent variable is normally distributed in both test and train, we can use standardization for doing feature scaling. Formula for feature scaling:

Standardized value = (Actual value – Mean)/ (standard deviation)

3. Model Building and Hyper parameters:

Algorithm used for Classification problem:

1) Logistic Regression
2) Naive Bayes
3) Decision tree
4) Random forest
5) XG Boost

3.1 Logistic Regression:

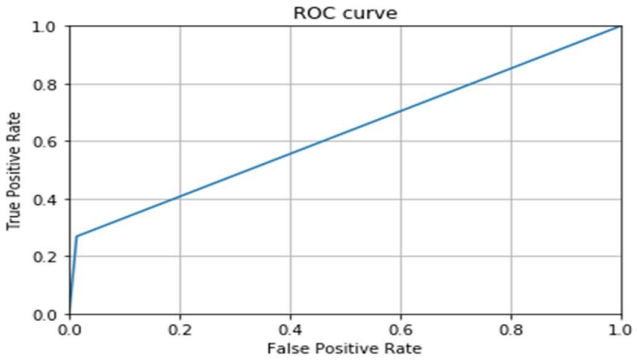
We have built a model using Logistic regression in python with all train data and in R using 80% train data. Initially used parameter:

Class_weight: “balanced”, this mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)))

Result in python: 'Accuracy':0.91439,'Precision':0.6905249679897567, 'Recall': 0.2683,'Specificity':0.9865

,

'False positive rate':0.01343, 'False negative rate':0.73166,'AUC': 0.627



Results in R: "accuracy 0.92","precision 0.69","recall 0.28","fpr 0.01","fnr 0.72", AUC 0.6321

In order to improve model to a satisfactory level we do hyper parameter tuning of model in python.
Tuned parameters:

1. penalty= [L1, L2]
   L1-It is basically minimizing the sum of the absolute differences (S) between the target value (Yi) and the estimated values (f(xi)):

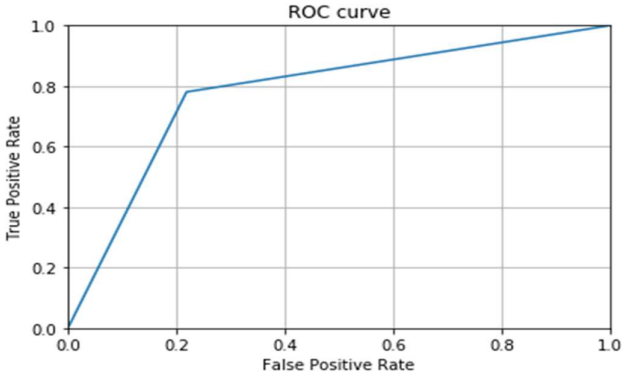
$$S = \sum_{a=1}^{n} |Y_a - f(x_a)|$$

   L2-It is basically minimizing the sum of the square of the differences (S) between the target value (Yi) and the estimated values (f(xi):

$$S = \sum_{a=1}^{n} (Y_a - f(x_a))^2$$

2. C: Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.
3. max_iter: number of iteration algorithms will run

Result in python after Hyper parameter tuning: 'Accuracy': 0.7810, 'Precision':0.2849,'Recall':0.7805, 'Specificity': 0.78114, 'False positive rate': 0.2188,'False negative rate':0.2194,'AUC':0.781.



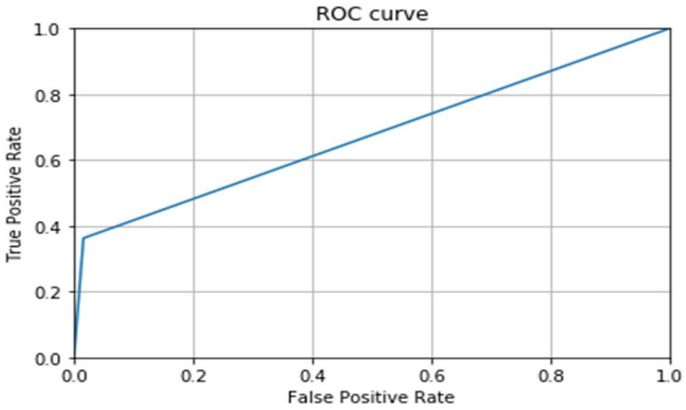
In order to improve model to a satisfactory level we use oversampled data in R in order to remove imbalancing in data.

Results in R: "accuracy 0.78", "precision 0.28", "recall 0.78","fpr 0.22","fnr 0.22", AUC 0.7798.

3.2 Naive Bayes:

We built a model with all default hyper parameter.

Results in Python: 'Accuracy':0.9216,'Precision':0.7181,'Recall':0.3624,'Specificity':0.9841,'False positive rate':0.01589,'False negative rate':0.6375,'AUC': 0.673



Results in R: "accuracy 0.92"," precision 0.71","recall 0.37", "fpr 0.02","fnr 0.63"," AUC 0.6745"
In order to improve model to a satisfactory level we do hyper parameter tuning of model in python.
Parameters:
Var_smoothing: Portion of the largest variance of all the features that is added to variances for calculation stability.

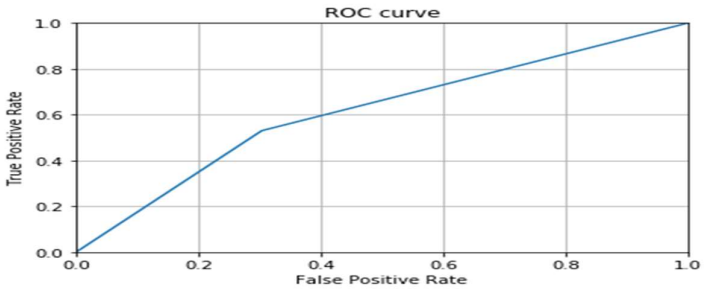
Result in python after hyper parameter tuning: No effect on results.
Results in R after using Over sampled data: "accuracy 0.81","precision 0.32","recall 0.8","fpr 0.19","fnr 0.2", AUC 0.8054.

3.3 Decision Tree:

After building model in python using all data and using decision tree algorithm, we find out that data is overfitting as all parameters like accuracy, precision, recall, specificity and ROC_auc is 1. In order to find out that model is over fitted or not we use cross validation using 10 folds, the results from the cross validation is show below: From this result we can see that model is overfitted.
test_accuracy 0.827365 test_precision 0.161610 test_recall 0.171261 test_roc_auc 0.535962
In order to make good model, we do hyper parameter tuning using these parameter

1. criterion: either 'gini' or 'entropy' based algo for decision tree
2. max_depth: max dept the tree will be made to grow
3. min_samples_split :specifies the minimum number of samples required to split an internal node

Results after hyper tuning: 'Accuracy':0.6796,'Precision':0.16325,'Recall':0.53025,'Specificity':0.6963,
'False positive rate':0.30362,'False negative rate': 0.4697,'AUC':0.613



Results in R with 80% train data: “accuracy 0.88","precision 0.26", "recall 0.12","fpr 0.74","fnr 0.09","AUC 0.5426"
Results in R after using Oversampled data: "accuracy 0.83","precision 0.17","recall 0.19","fpr 0.83","fnr 0.09","AUC:0.5451"

3.4 Random Forest:

We have built only one model of random forest using all the data in python. After observing the results, we can say that model is overfitted as as all parameters like accuracy, precision, recall, specificity and ROC_auc is 1. In order to find out that model is over fitted or not we use cross validation using 10 folds, the results from the cross validation is show below: From this result we can see that model is overfitted.

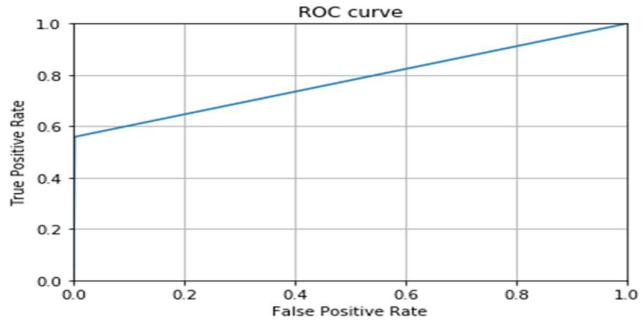
```
test_accuracy 0.899535 test_precision 0.500000 test_recall 0.000249
test_roc_auc 0.821880
```

We have written the codes for hyper parameter tuning Random forest in python.
(Do not using random forest in R as it takes so much time and memory)

3.5 XG Boost:

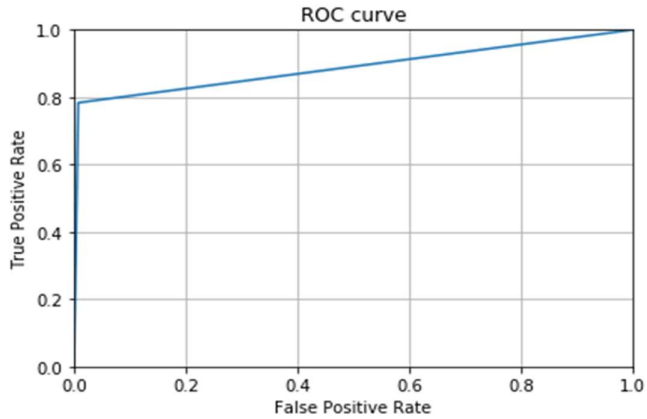
XG boost is modify version of Gradient boosted machine.

We have built a model on python using XG boost on all the provided data, the results we get are: 'Accuracy':0.95405,'Precision':0.97270,'Recall':0.55846,'Specificity':0.99824 ,'False positive rate':0.001750,'False negative rate':0.4415, 'AUC':0.778



After hyper parameter tuning of XG boost, Parameter used:
1. Learning_rate:Makes the model more robust by shrinking the weights on each step
2. N_estimators: Number of trees to create
3. Max_depth: Max depth tree is allowed to grow
4. Min_child_weight: Defines the minimum sum of weights of all observations required in a child.
5. Gamma: A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.
6. Subsample: Same as the subsample of GBM. Denotes the fraction of observations to be randomly samples for each tree.
7. Colsample_bytree: Similar to max_features in GBM. Denotes the fraction of columns to be randomly samples for each tree.
8. Objective: This defines the loss function to be minimized.
9. Scale_pos_weight: A value greater than 0 should be used in case of high-class imbalance as it helps in faster convergence.

Results after hyper parameter tuning: 'Accuracy':0.971615,'Precision':0.92228, 'Recall':0.78356, 'Specificity':0.99262,'False positive rate':0.00737, 'False negative rate':0.21643,'AUC':0.888.



In R we built model with 80 % training data and oversampled data using XG boost. Results are written below:
Results with over sampled data: “accuracy 0.71","precision 0.24","recall 0.88","fpr 0.76","fnr 0.02","AUC 0.7867"
Results with provided data: "accuracy 0.92","precision 0.7","recall 0.34","fpr 0.3","fnr 0.07","AUC 0.6621"

4.Error Matrix:

Confusion Matrix:
The confusion matrix shows the ways in which your classification model is confused when it makes predictions.

	Predicted class 0	Predicted class 1
Actual class 0	TN	FP
Actual class 1	FN	TP

Accuracy: (TN+TP)/ (TN+TP+FN+FP)

Precision: This indicated percentage of predicted positive cases which is actually positive.
P= (TP)/(TP+FP)
Recall: This indicated how much percentage actual positive cases are predicted as positive.
R=(TP)/(TP+FN)
Specificity: This indicated percentage of actual negative cases which is predicted as negative.
S= (TN)/(TN+FP)
ROC-AUC: ROC curve is a performance measurement for classification problem at various thresholds settings. This parameter represents degree or measure of separability. It tells how much model is capable of distinguishing between classes.

5.Model selection:

As this is imbalancing problem we cannot select model just on base of accuracy. So, we should have to select the model which have comparatively high ROC_AUC as this model is more capable to differentiated

classes. Apart from that model must have precision and recall both as possible as near to 1. Also, False negative rate and false positive rate should be low.

Result from Python:

	Accuracy	Precision	Recall	Specificity	False positive rate	False negative rate	AUC
XGBH	0.971615	0.922280	0.78356	0.99262	0.00737	0.21643	0.888
LRH	0.781000	0.284900	0.78050	0.78114	0.21880	0.21940	0.781
XGB	0.954050	0.972700	0.55846	0.99824	0.00175	0.44150	0.778
NB	0.921600	0.718100	0.36240	0.98410	0.01589	0.63750	0.673
NBH	0.921600	0.718100	0.36240	0.98410	0.01589	0.63750	0.673
LR	0.914390	0.690525	0.26830	0.98650	0.01343	0.73166	0.627
DTH	0.679600	0.163250	0.53025	0.69630	0.30362	0.46970	0.613

The Model we built using XG boost after we have done hyper parameter tuning best from all the models i.e. model XG_model_hyper (XGBH).

Result from R:

#XG_pred_over:"accuracy 0.71","precision 0.24","recall 0.88","fpr 0.76","fnr 0.02","AUC 0.7867"

#XG_pred:"accuracy 0.92","precision 0.7","recall 0.34","fpr 0.3","fnr 0.07","AUC 0.6621"

#DT_pred_over:"accuracy 0.83","precision 0.17","recall 0.19","fpr 0.83","fnr 0.09","AUC:0.5451"

#DT_pred:"accuracy 0.88","precision 0.26", "recall 0.12","fpr 0.74","fnr 0.09","AUC 0.5426"

#NB_pred_over:"accuracy 0.81","precision 0.32","recall 0.8","fpr 0.19","fnr 0.2", AUC 0.8054

#NB_pred:"accuracy 0.92", precision 0.71","recall 0.37", "fpr 0.02","fnr 0.63", AUC 0.6745

#LB_pred_over: "accuracy 0.78", "precision 0.28", "recall 0.78","fpr 0.22","fnr 0.22", AUC 0.7798

#logit_pred:"accuracy 0.92","precision 0.69","recall 0.28","fpr 0.01","fnr 0.72", AUC 0.6321

The model we built using Naïve Bayes when we are using over sampled data is beast from all the models i.e. model NB_model_over.