Project Report

Santander Customer Transaction Prediction

By: Krishna.R

1. INTRODUCTION

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?

1.1. Data

Dataset is provided in a csv file train.csv and test.csv.

The data provided is anonymized dataset containing numeric feature variables, the binary target column, and a string ID_code column. The task is to predict the value of target column in the test set.

1.2. Steps Preformed in this Project

The whole project is divided in 7 phases (and further subphases). Below are the phases defined:

- Define problem statement
- Gather the data
- Prepare the data
- Perform Exploratory Data Analysis
- Feature Scaling
- Feature Selection
- Dealing with imbalanced data
- Modeling
- Evaluate and compare model performance and choose the best model

2. 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.

3. Gather the data

The data is provided in csv file. Both train and test dataset is provided separately. All we have to do is import the data and start working.

Let's Import the data,

```
%%time
# Importing train and test dataset
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

Both train and test dataset is imported, now let's quickly understand both train and test data.

First let's look at train dataset

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...

Let's check the dimension of the train dataset

```
train.shape
```

```
(200000, 202)
```

There are 200,000 observations and 202 columns in the train columns.

Let's check datatypes of each columns in train dataset

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Columns: 202 entries, ID_code to var_199
dtypes: float64(200), int64(1), object(1)
memory usage: 308.2+ MB
```

There are 200 columns with float datatype, one columns with object datatype i.e. "ID_code" and "target" column with integer datatypes.

Now let's look at test dataset

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	...
0	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	...
1	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	...
2	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	...
3	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	...
4	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	...

Let's check the dimension of the test dataset

```
test.shape
```

```
(200000, 201)
```

There are 200,000 observation and 201 columns in test dataset

Let's check datatypes of the each column in test dataset

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Columns: 201 entries, ID_code to var_199
dtypes: float64(200), object(1)
memory usage: 306.7+ MB
```

In test dataset there are 200 columns with float datatype and one column with object datatype i.e. "ID_code".

4. Data Preprocessing

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

4.1. Missing Value Analysis

Missing values occur when no data value is stored for the variable in an observation. Missing data are a common occurrence and can have a significant effect on the conclusions that can be drawn from the data. Let's check if missing values are present in train and test dataset.

```
train.isnull().values.any()
```

```
False
```

We can see the train dataset has no missing values, let's also check in test dataset

```
test.isnull().values.any()
```

```
False
```

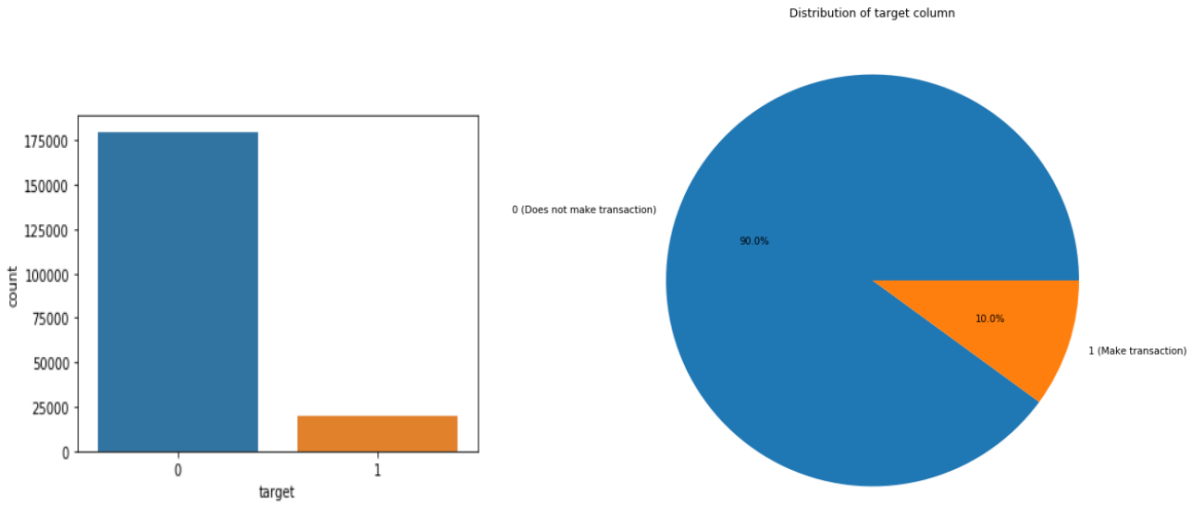
Test dataset also has no missing values. So we can proceed with further analysis.

5. Exploratory Data Analysis

Now let's explore "target" variable.

Distribution of "target" variable

```
0    179902
1     20098
Name: target, dtype: int64
```



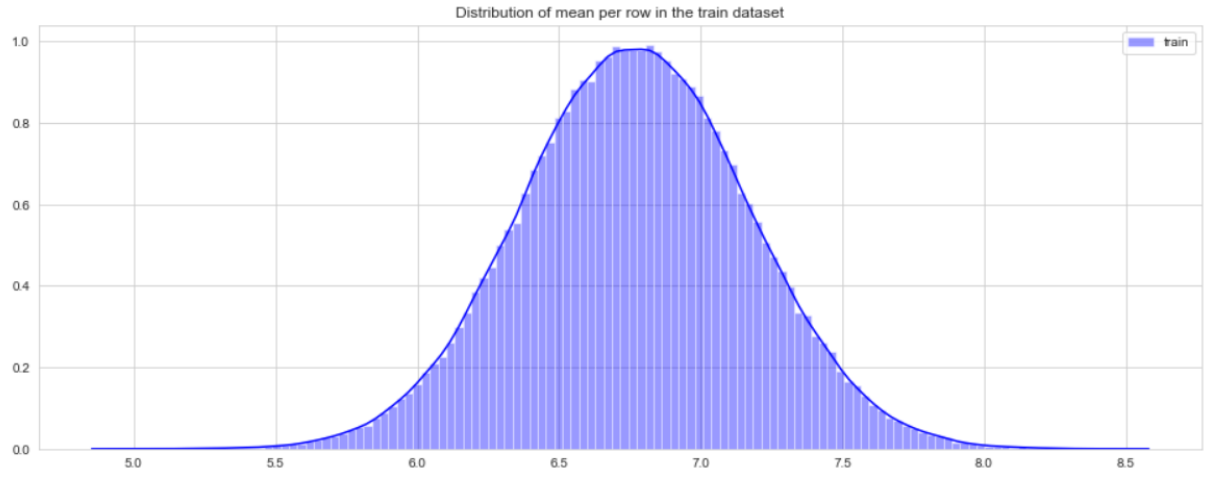
Out of 200,000 observation there 179902 "0" and 20098 "1" in the target variable. "0" is distributed 90% and "1" is distributed on 10%, So we are dealing with highly imbalanced data. We also learn that only 90% percent of the customers did not make transaction and only 10% made transaction. Now let's convert the target column from integer to categorical.

```
# Converting target column from numeric to categorical
train['target'] = train['target'].astype('category')
```

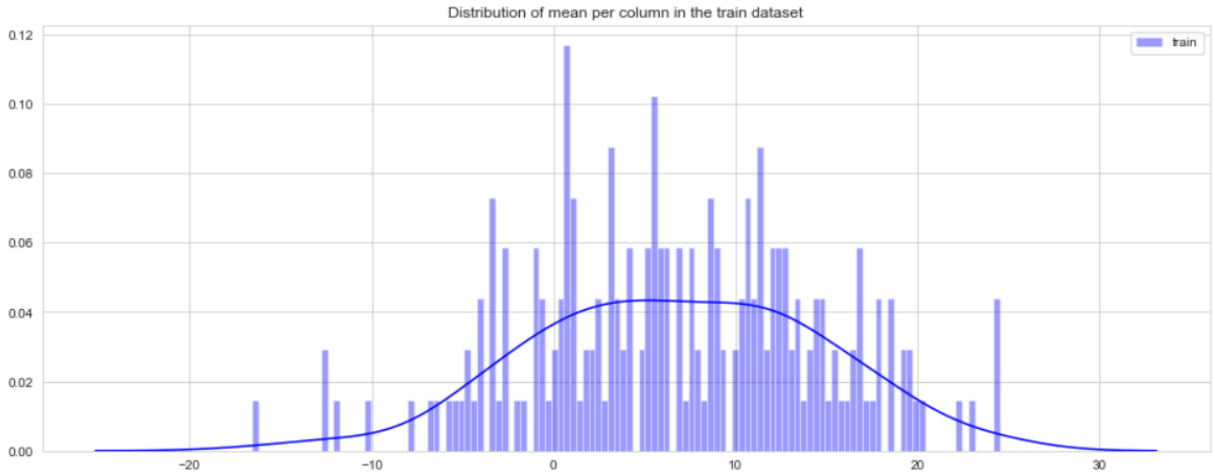
Distribution of each numeric features:

Almost all the features follow normalized distribution.

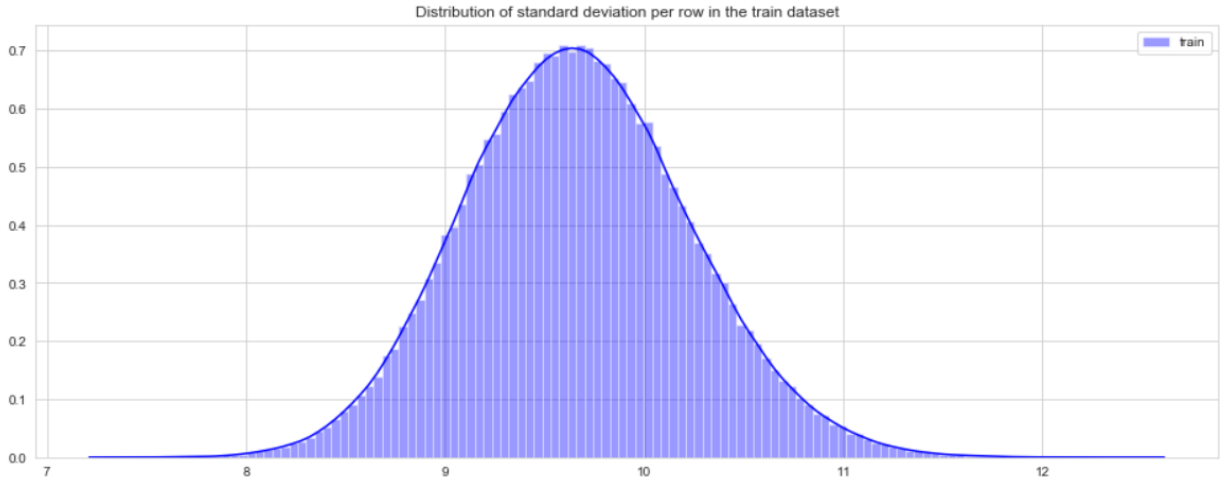
Distribution of mean values per row in train dataset:



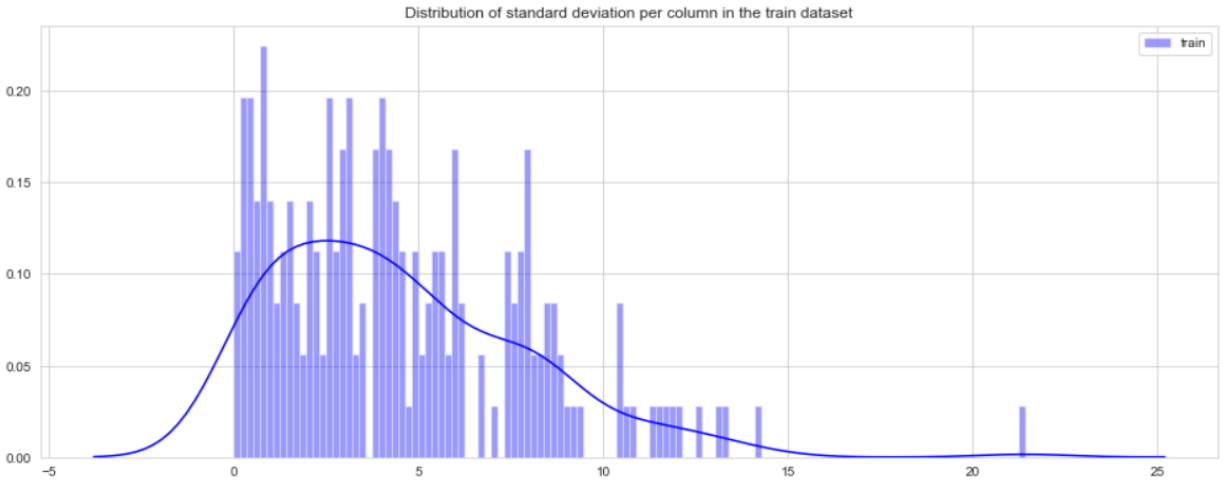
Distribution of mean per columns in train dataset:



Distribution of standard deviation per row in train dataset:

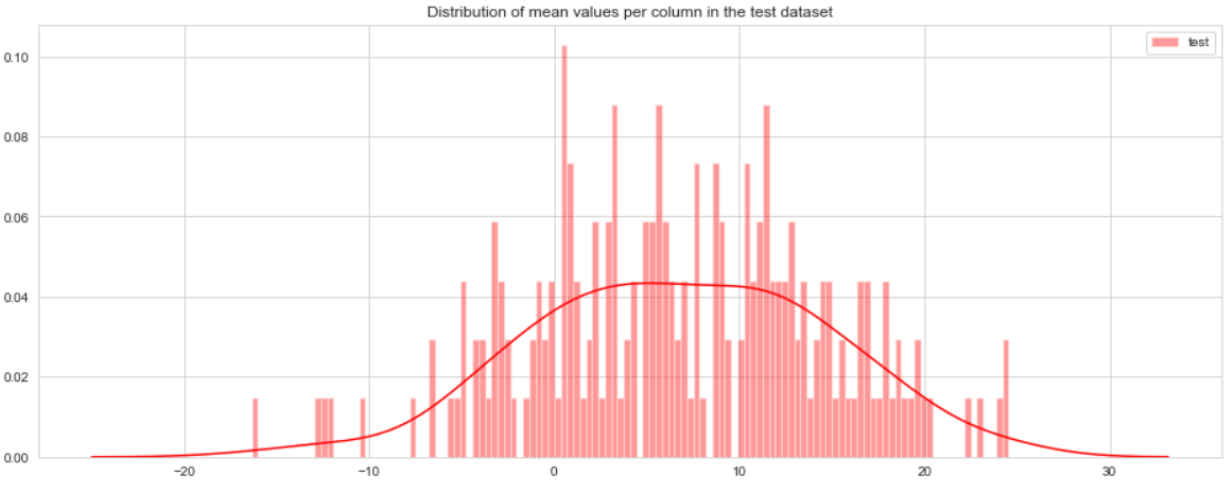


Distribution of standard deviation per column in train dataset:

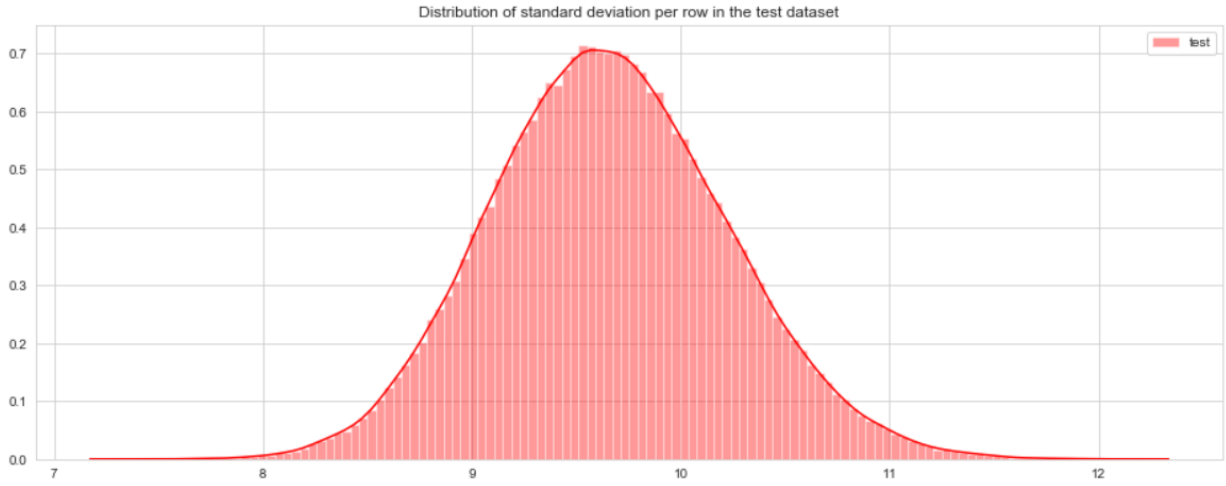


Distribution of mean per row in test dataset:

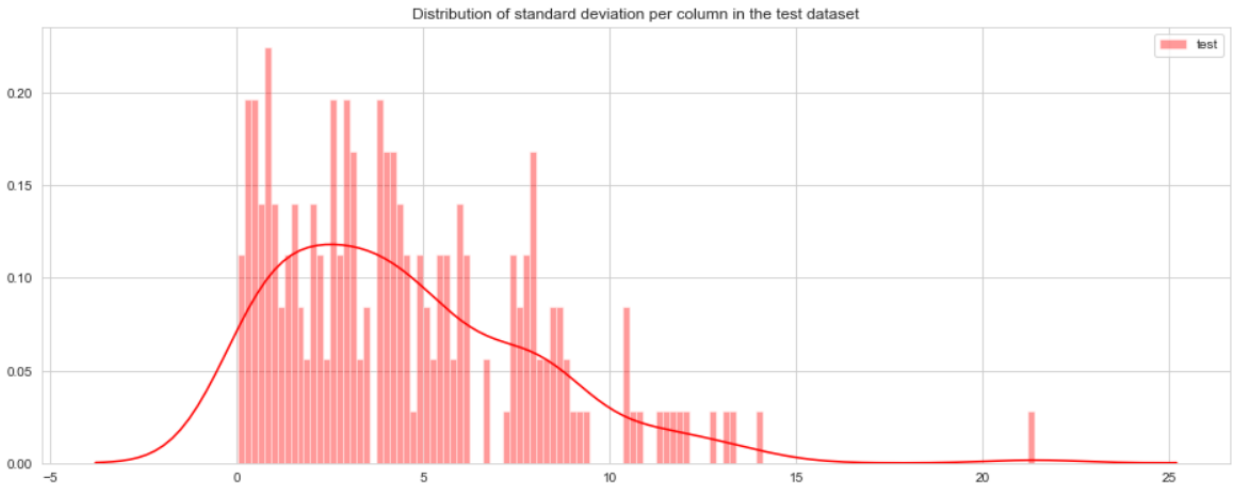
Distribution of mean per column in the test dataset:



Distribution of standard deviation per row in test dataset:



Distribution of standard deviation per column in test dataset:



6. Feature Scaling

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. Feature scaling through standardization (or Z-score normalization) can be an important preprocessing step for many machine learning algorithms. Standardization involves rescaling the features such that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one.

Standardization (Z-score normalization):

Standardization (or Z-score normalization) is the process of rescaling the features so that they'll have the properties of a Gaussian distribution with μ=0 and σ=1. Formula for z-score is,

$$z = \frac{x_i - \mu}{\sigma}$$

Standardization of train and test data using StandarScaler() function:

```
train_scaled = pd.DataFrame(StandardScaler().fit_transform(train_features), columns = train_features.columns)
test_scaled = pd.DataFrame(StandardScaler().fit_transform(test_features), columns = test_features.columns)
```

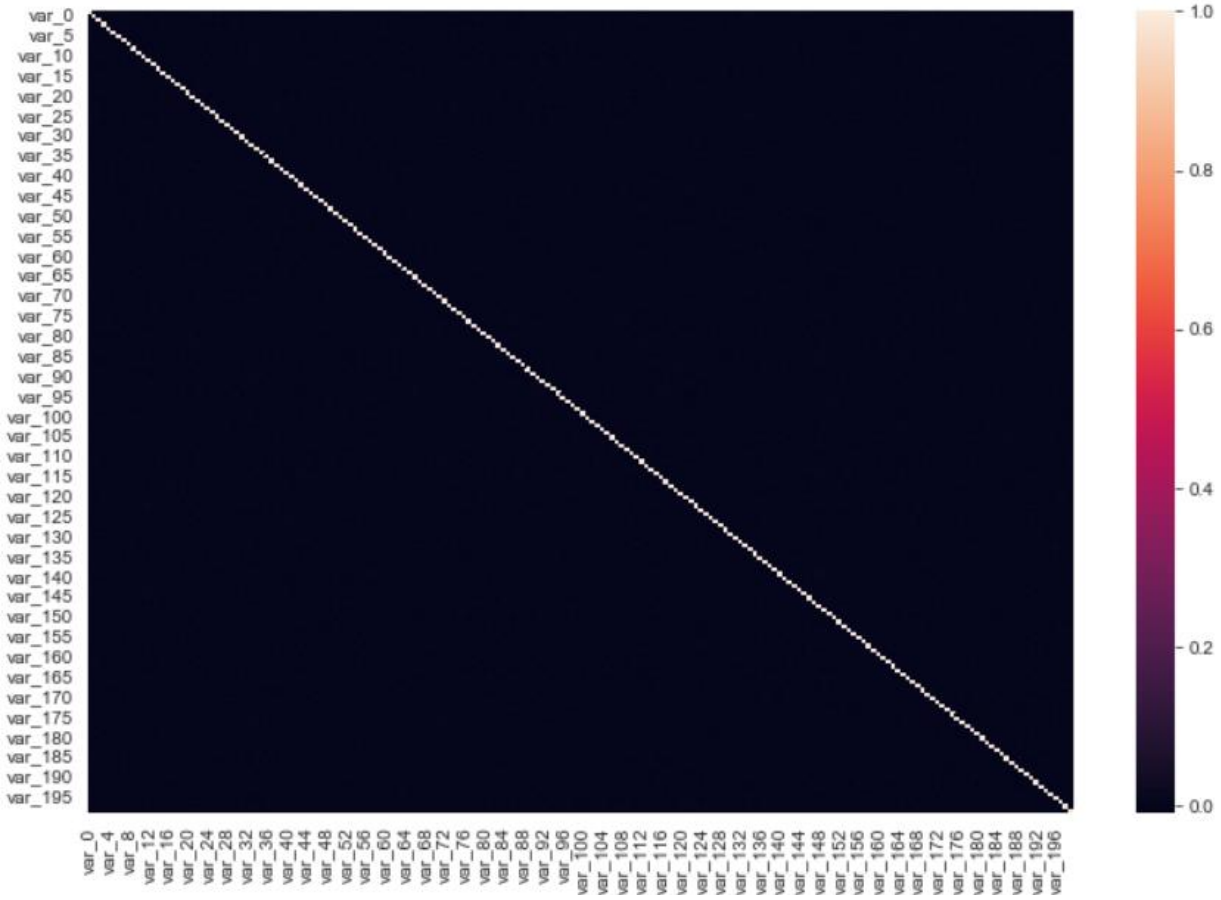
7. Feature Selection

Feature Selection is also called variable selection and attribute selection … is the process of selecting a subset of relevant features for use in model construction.

Feature selection is different from dimensionality reduction. Both methods seek to reduce the number of attributes in the dataset, but a dimensionality reduction method do so by creating new combinations of attributes, whereas feature selection methods include and exclude attributes present in the data without changing them.

7.1. Correlation plot on numerical features

Correlation plot on numerical features to look for highly correlated features



From the correlation plot above we can clearly see there are no highly correlated features.

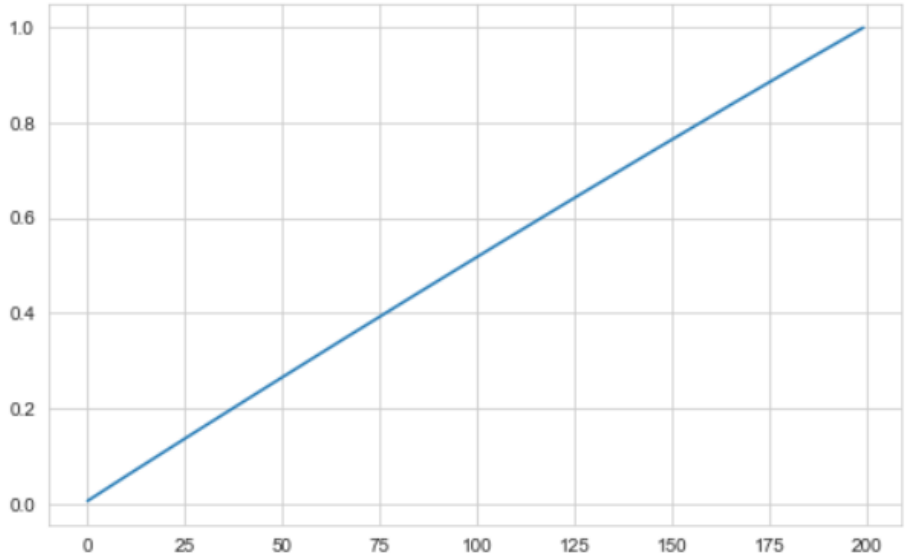
7.2. Principle Component Analysis

Principal component analysis (PCA) is a technique used for identification of a smaller number of uncorrelated variables known as principal components from a larger set of data. The technique is widely used to emphasize variation and capture strong patterns in a data set. Invented by Karl Pearson in 1901, principal component analysis is a tool used in predictive models and exploratory data analysis.

How does PCA work:

1. Calculate the covariance matrix X of data points.
2. Calculate eigen vectors and corresponding eigen values.
3. Sort the eigen vectors according to their eigen values in decreasing order.
4. Choose first k eigen vectors and that will be the new k dimensions.
5. Transform the original n dimensional data points into k dimensions.

Plotting the explained variance ratio after performing Principle component analysis on the data:



The line of cumulative sums of explained variance ratio after PCA on the data is indicating that the data has already undergone PCA (get straight line i.e. y=x). So we have to go with all 200 features.

8. Dealing with imbalanced data

Before modelling for this dataset let us understand how to deal with imbalanced dataset for classification problem. Traditional Machine Learning algorithms tend to produce unsatisfactory classifiers when faced with imbalanced datasets. For any imbalanced data set, if the event to be predicted belongs to the minority class and the event rate is less than 10%, it is usually referred to as a rare event. The conventional model evaluation methods do not accurately measure model performance when faced with imbalanced datasets. Standard classifier algorithms like Decision Tree and Logistic Regression have a bias towards classes which have large number of instances. They tend to only predict the majority class data. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class. Evaluation of a classification algorithm performance is measured by the Confusion Matrix which contains information about the actual and the predicted class.

Actual	Predicted	
	Positive Class	Negative Class
Positive Class	True Positive(TP)	False Negative (FN)
Negative Class	False Positive (FP)	True Negative (TN)

Accuracy of a model = (TP+TN) / (TP+FN+FP+TN)

However, while working in an imbalanced domain accuracy is not an appropriate measure to evaluate model performance. Hence, we need evaluation metrics such as Recall, Precision, F1_score (harmonic mean of Precision and recall), AUC-ROC score along with Accuracy.

How to deal with these imbalanced datasets?

- Using Resampling techniques such as Random under Sampling, Random Over Sampling, Cluster Based Oversampling, Informed over Sampling (synthetic Minority Over Sampling Technique) are useful.
- Using Ensemble techniques like bagging based ensembling or boosting based ensembling (Adaptive Boosting, Gradient Tree Boosting, Extreme Gradient Boosting)
- Changing Performance metrics for model evaluation
- Changing machine learning algorithm.

Techniques:

1. Random Under Sampling

Random Under sampling aims to balance class distribution by randomly eliminating majority class examples. This is done until the majority and minority class instances are balanced out.

Advantages

- It can help improve run time and storage problems by reducing the number of training data samples when the training data set is huge.

Disadvantages

- It can discard potentially useful information which could be important for building rule classifiers.
- The sample chosen by random under sampling may be a biased sample. And it will not be an accurate representation of the population. Thereby, resulting in inaccurate results with the actual test data set.

2. Random Over Sampling

Over-Sampling increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample.

Advantages

- Unlike under sampling this method leads to no information loss.
- Outperforms under sampling

Disadvantages

- It increases the likelihood of overfitting since it replicates the minority class events.

3. Cluster-Based Over Sampling

In this case, the K-means clustering algorithm is independently applied to minority and majority class instances. This is to identify clusters in the dataset. Subsequently, each cluster is oversampled such that all clusters of the same class have an equal number of instances and all classes have the same size.

Advantages

- This clustering technique helps overcome the challenge between class imbalance. Where the number of examples representing positive class differs from the number of examples representing a negative class.
- Also, overcome challenges within class imbalance, where a class is composed of different sub clusters. And each sub cluster does not contain the same number of examples.

Disadvantages

- The main drawback of this algorithm, like most oversampling techniques is the possibility of over-fitting the training data.

4. Informed Over Sampling: Synthetic Minority Over-Sampling Technique

This technique is followed to avoid overfitting which occurs when exact replicas of minority instances are added to the main dataset. A subset of data is taken from the minority class as an example and then new synthetic similar instances are created. These synthetic instances are then added to the original dataset. The new dataset is used as a sample to train the classification models.

Advantages

- Mitigates the problem of overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances
- No loss of useful information

Disadvantages

- While generating synthetic examples SMOTE does not take into consideration neighboring examples from other classes. This can result in increase in overlapping of classes and can introduce additional noise
- SMOTE is not very effective for high dimensional data

5. Bagging Based Ensembling

Bagging is an abbreviation of Bootstrap Aggregating. The conventional bagging algorithm involves generating ‘n’ different bootstrap training samples with replacement. And training the algorithm on each bootstrapped algorithm separately and then aggregating the predictions at the end.

Bagging is used for reducing Overfitting in order to create strong learners for generating accurate predictions. Unlike boosting, bagging allows replacement in the bootstrapped sample.

Advantages

- Improves stability & accuracy of machine learning algorithms
- Reduces variance
- Overcomes overfitting
- Improved misclassification rate of the bagged classifier
- In noisy data environments bagging outperforms boosting

Disadvantages

- Bagging works only if the base classifiers are not bad to begin with. Bagging bad classifiers can further degrade performance

6. Boosting-Based Ensembling

Boosting is an ensemble technique to combine weak learners to create a strong learner that can make accurate predictions. Boosting starts out with a base classifier / weak classifier that is prepared on the training data.

The base learners / Classifiers are weak learners i.e. the prediction accuracy is only slightly better than average. A classifier learning algorithm is said to be weak when small changes in data induce big changes in the classification model.

In the next iteration, the new classifier focuses on or places more weight to those cases which were incorrectly classified in the last round.

If we use sampling then we will be increasing observations (already we have 200000 observations) and hence speed will be lower. Also useful techniques such as SMOTE are susceptible to outliers and not fit for high dimensional data (we have 410 features). Other sampling techniques can cause overfitting also. Hence I am not using sampling techniques.

9. Modelling

Algorithm used for modelling are:

- Logistics Regression
- Decision Tree
- Random Forest
- XGboost
- LightGBM

Logistic Regression

is the appropriate regression analysis to conduct when the dependent variable is binary. Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Decision Tree

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

Random Forest

Random forests, also known as random decision forests, are a popular ensemble method that can be used to build predictive models for both classification and regression problems. Ensemble methods use multiple learning models to gain better predictive results — in the case of a random forest, the model creates an entire forest of random uncorrelated decision trees to arrive at the best possible answer.

XGboost

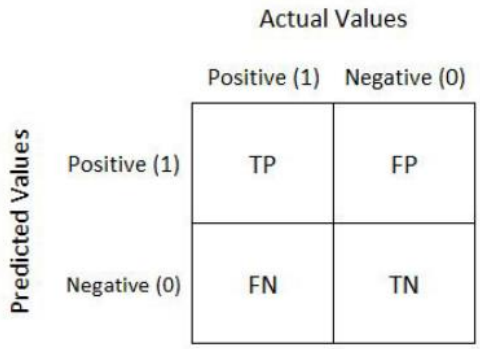
XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree-based algorithms are considered best-in-class right now.

LightGBM

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 and Capable of handling large-scale data.

Metrics

This is a classification problem and we need to understand confusion matrix for getting evaluation metrics. It is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values. It is extremely useful for measuring Recall, Precision, Specificity, Accuracy and most importantly AUC-ROC Curve.



True Positive: You predicted positive and it's true.

True Negative: You predicted negative and it's true.

False Positive: (Type 1 Error) You predicted positive and it's false.

False Negative: (Type 2 Error) You predicted negative and it's false.

Based on confusion matrix we have following evaluation metrics:

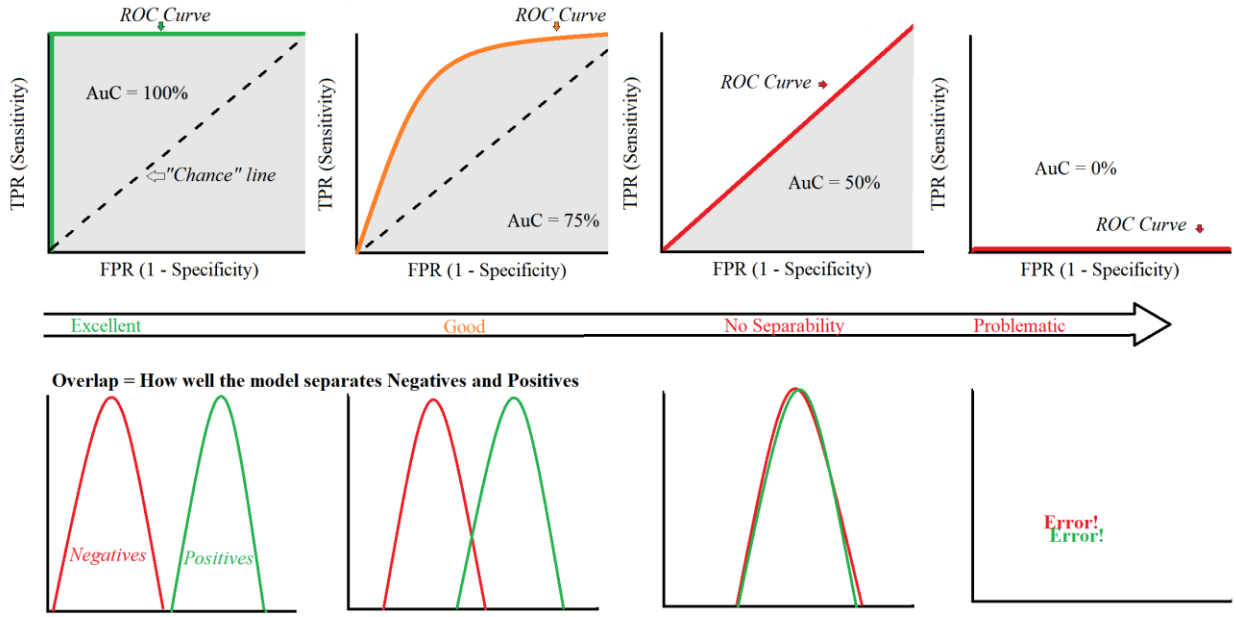
Recall - Out of all the positive classes, how much we predicted correctly. It should be high as possible.

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

F-measure- 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.

AUC-ROC Score - It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (Area Under the Receiver Operating Characteristics) AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes.

Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.



An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity.

Following are the metrics evaluated for different classification models.

	Model	AUC	Accuracy	Precision	Recall	f1_score
0	Logistic Regression	0.630654	0.914433	0.684384	0.275502	0.392857
1	Decision Tree	0.557966	0.835750	0.199308	0.210317	0.204665
2	Random Forest	0.506999	0.899783	0.547059	0.015425	0.030005
3	XBboost	0.507961	0.900983	0.915094	0.016089	0.031622
4	LightGBM	0.794508	0.869400	0.411914	0.700780	0.518851

Model selected to predict on test data: LightGBM

Reason for acceptance: Even though Logistic Regression has better accuracy than LightGBM, AUC is considered better metric when it comes to binary classification. AUC also tells us how well

the model is at separability of the classes. Higher the value of AUC better the model is. Therefore, AUC score is the main priority.

Reason for rejection of other models: I had to reject Logistic Regression because it's auc, recall and f1 score is lower compared to LightGMB. AUC score for LightGBM is 0.794 which is pretty good score and It tells that the model is good at separating classes

Decision tree is rejected because every metric i.e. auc, accuracy, precision, recall and f1 score is lower than that of LightGBM. Also, AUC score is 0.557 that means the model is bad at separating classes

Random forest is rejected because auc, recall and f1 score is lower than that of LightGBM. AUC score for random forest is 0.506 that means the model is bad at separating classes

XGboost is rejected because auc, recall and f1 score is lower than that of LightGBM. AUC for XGboost is 0.507 that means the model is bad at separating classes

Tuning LightGBM Model

Parameter Tuning:

Following set of practices can be used to improve your model efficiency

1. **num_leaves**: This is the main parameter to control the complexity of the tree model. Ideally, the value of num_leaves should be less than or equal to 2^(max_depth). Value more than this will result in overfitting.
2. **min_data_in_leaf**: Setting it to a large value can avoid growing too deep a tree, but may cause under-fitting. In practice, setting it to hundreds or thousands is enough for a large dataset.
3. **max_depth**: You also can use max_depth to limit the tree depth explicitly.

For Faster Speed:

- Use bagging by setting bagging_fraction and bagging_freq
- Use feature sub-sampling by setting feature_fraction
- Use small max_bin
- Use save_binary to speed up data loading in future learning
- Use parallel learning,

For better accuracy:

- Use large max_bin (may be slower)
- Use small learning_rate with large num_iterations
- Use large num_leaves(may cause overfitting)
- Use bigger training data
- Try dart
- Try to use categorical feature directly

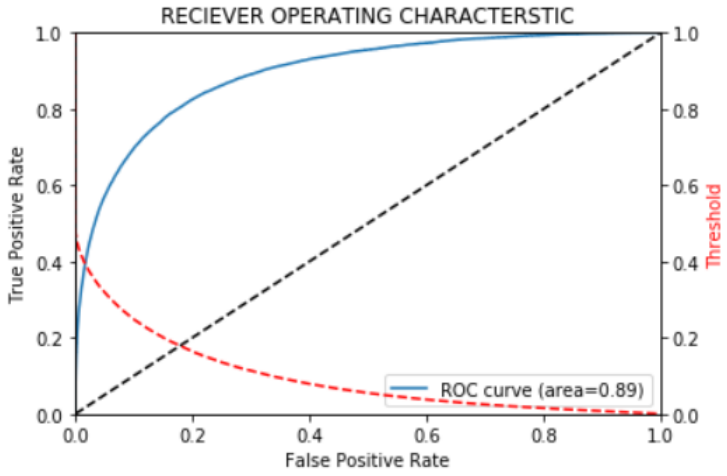
To deal with overfitting:

- Use small max_bin
- Use small num_leaves
- Use min_data_in_leaf and min_sum_hessian_in_leaf
- Use bagging by set bagging_fraction and bagging_freq
- Use feature sub-sampling by set feature_fraction
- Use bigger training data
- Try lambda_l1, lambda_l2 and min_gain_to_split to regularization
- Try max_depth to avoid growing deep tree

After tuning LightGBM model with appropriate parameters we get following results.

```
Accuracy Score for LightGBM is  0.880345
Precision Score for  LightGBM is  0.6936013533684944
Recall Score for LightGBM is  0.43956736984832717
AUC Score for LightGBM is  0.8948618893966243
f1 Score for LightGBM is  0.5381096678311555
```

AUC-ROC Curve:



Code:

Code for Python and R is provided in .ipynb and R file respectively.

To Run Python notebook Python and Jupyter notebook should be installed

To run R, R Language should be installed.