Customer Transaction Prediction

25.12.2019

Manish Manik Apte

Table of Contents

Background -

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

Problem Statement -

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 Details -

Provided with an anonymized dataset containing 200 numeric feature variables, the binary target column, and a string ID_code column.

Problem Analysis -

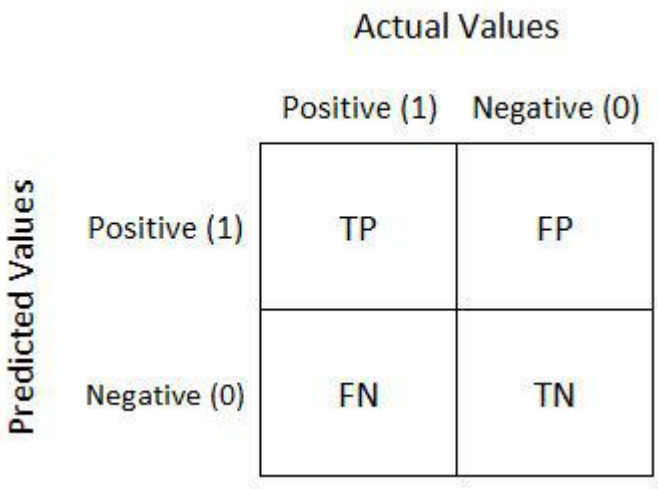
This is a binary classification problem under supervised machine learning algorithm.
The task is to predict the value of target column in the test set.

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.

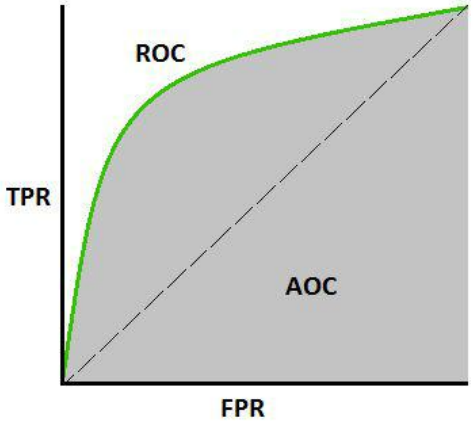
F-measure- It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

Another important measure is 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.



$$\text{TPR / Recall / Sensitivity} = \frac{TP}{TP + FN} \qquad \text{Specificity} = \frac{TN}{TN + FP} \qquad \text{FPR} = 1 - \text{Specificity} = \frac{FP}{TN + FP}$$

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.

Exploratory Data Analysis

Check for variable data types in train and test data.

Id_code is object type, target is int type and 200 anonymous variables of float type.

Check for missing values

No missing values.

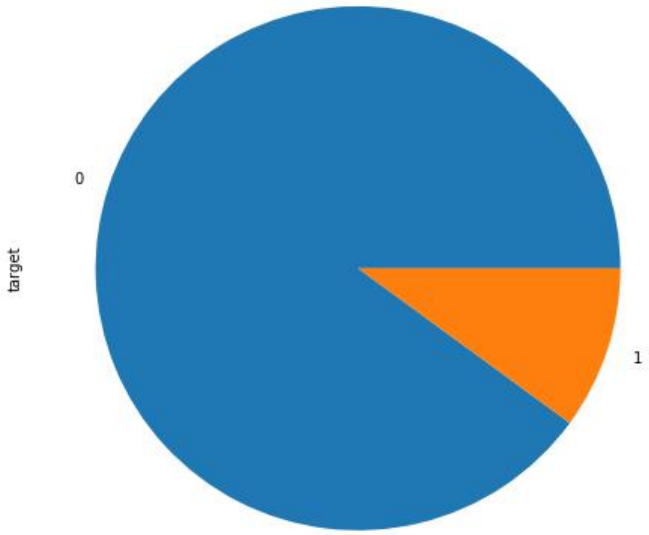
Shape of data

200000 observations with 202 columns in train data and 200000 observations with 201 columns in test data.

Check Balance of target column

0 179902
1 20098 Imbalanced

Dataset.



Visualizations

Distribution of columns

See image (attached with submission) distribution of columns.png

Almost all features follow normalised distribution.

Distribution of all numerical features per each class.

See image (attached with submission) Distribution of columns per each class.

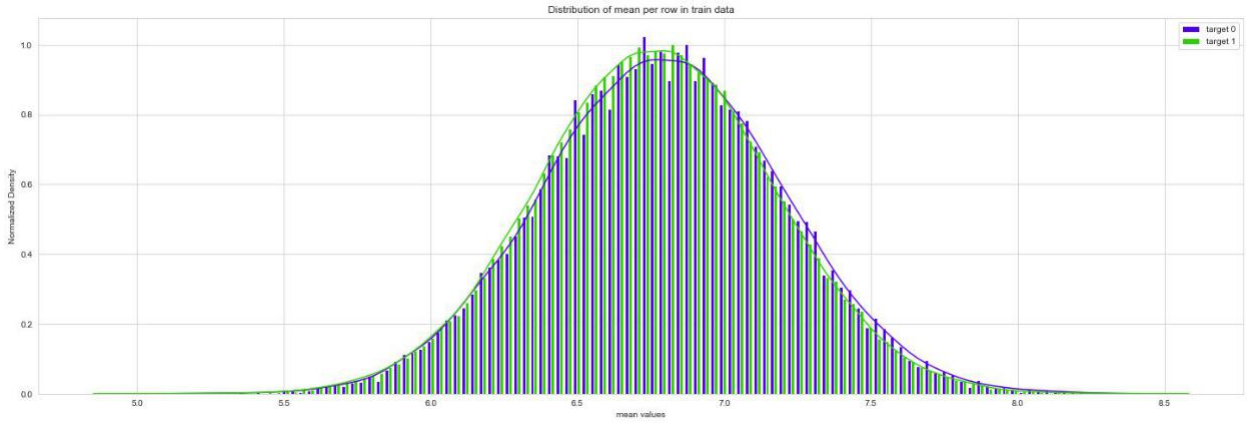
We can observe that there is a considerable number of features with significant different distribution for the two target values. For example, var_0, var_1, var_2, var_5, var_9, var_13, var_106, var_109, var_139 and many others. Also some features, like var_2, var_13, var_26, var_55, var_175, var_184, var_196 shows a distribution that resembles a bivariate distribution.

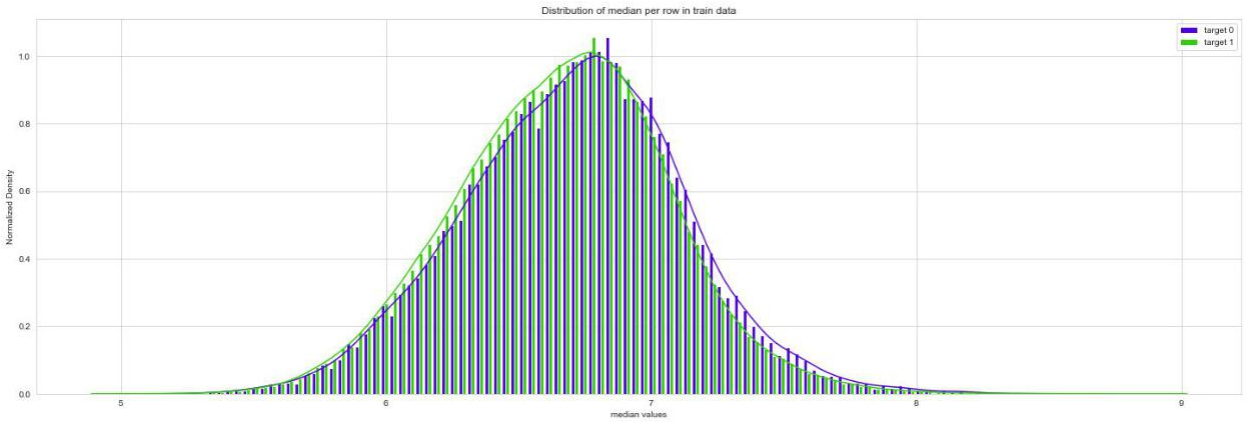
Distribution of numerical variables in train and test data.

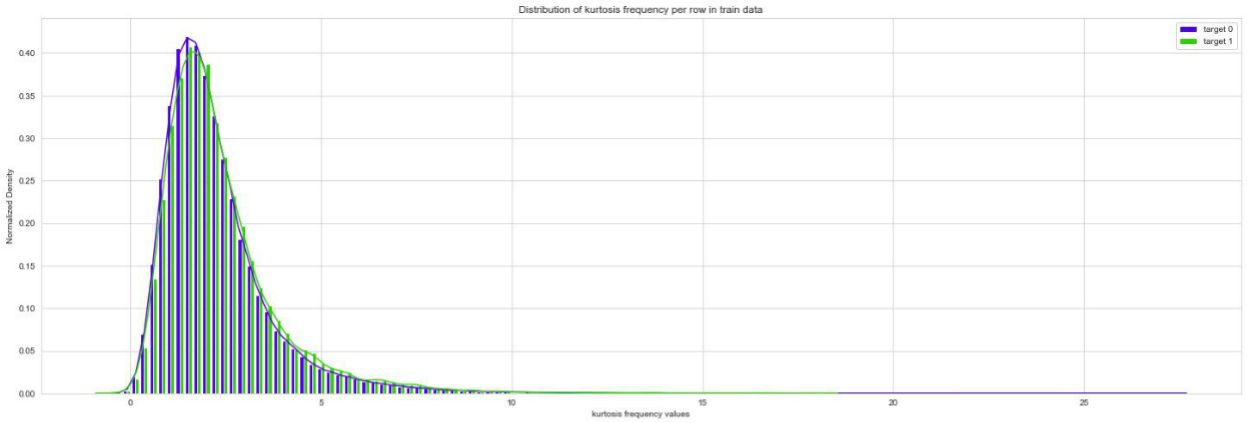
See image (attached with submission) distribution of train and test data.png

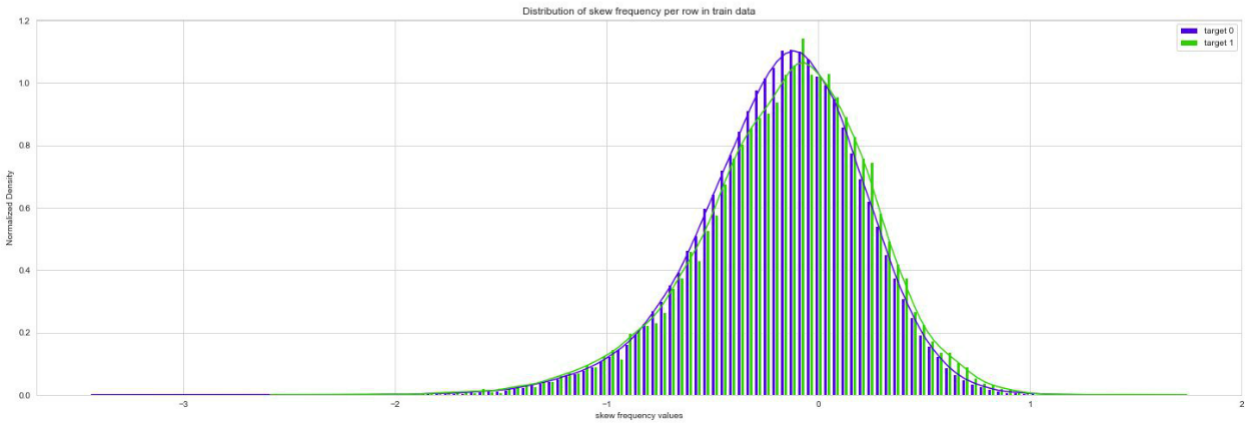
The train and test seems to be well balanced with respect to distribution of the numeric variables.

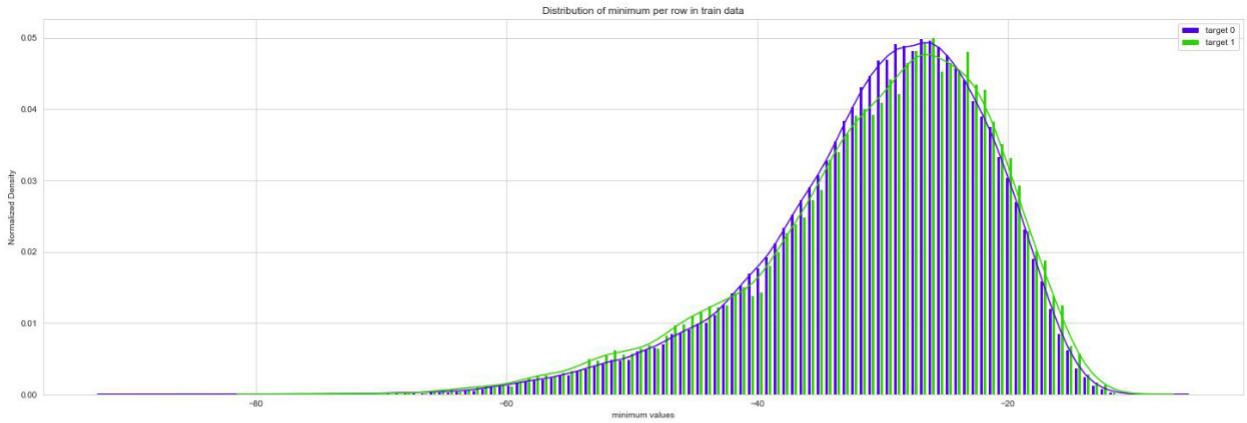
Row Wise Distribution of mean, median, standard deviation, kurtosis, skewness, min, max of train dataset for both targets (0 and 1)

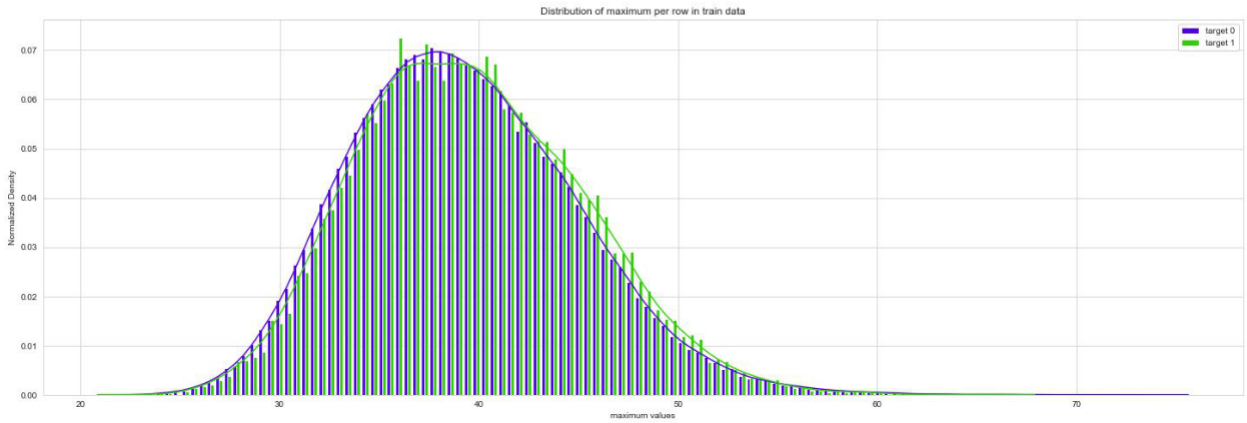



Distribution of median per row in train data
target 0
target 1
Normalized Density
median values

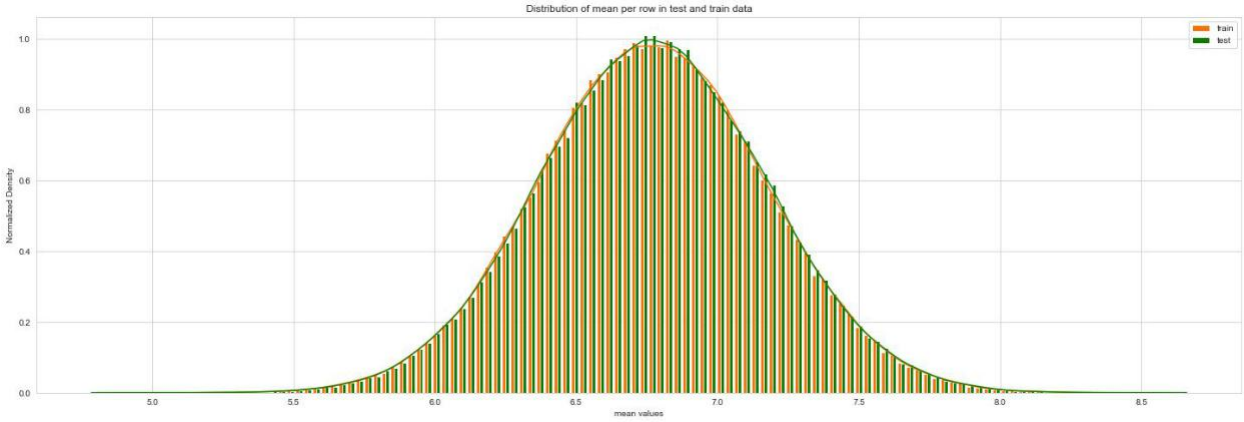

Distribution of kurtosis frequency per row in train data
target 0
target 1
Normalized Density
kurtosis frequency values

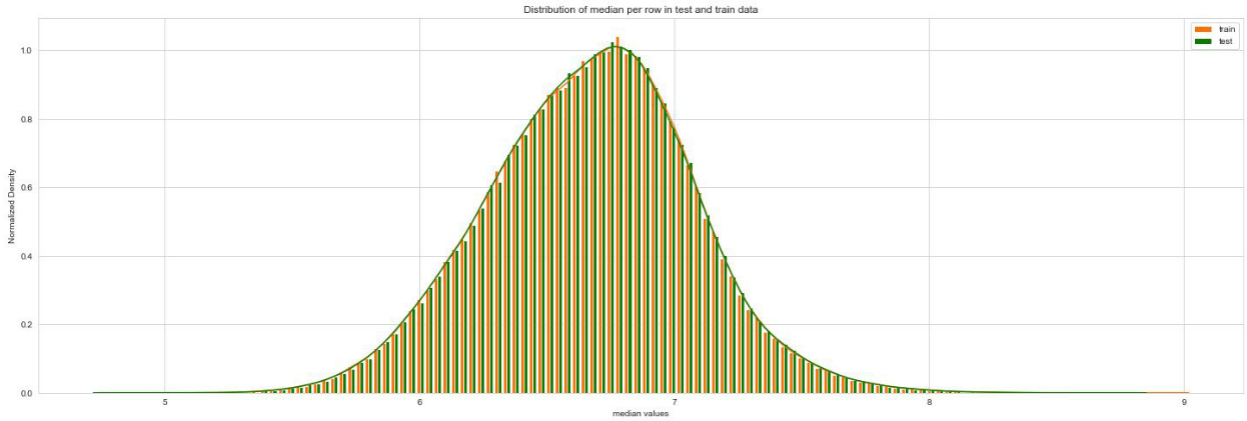

Distribution of skew frequency per row in train data
target 0
target 1
Normalized Density
skew frequency values

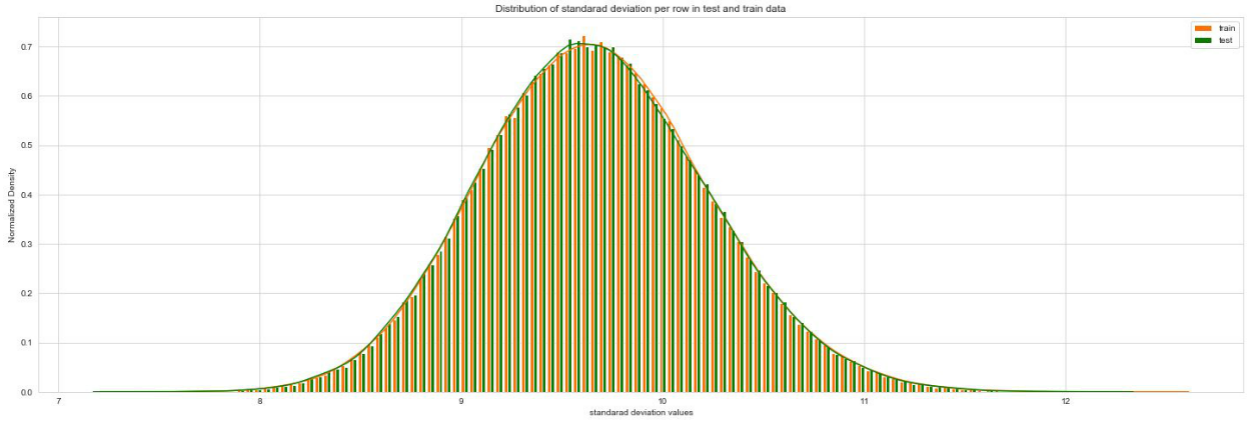


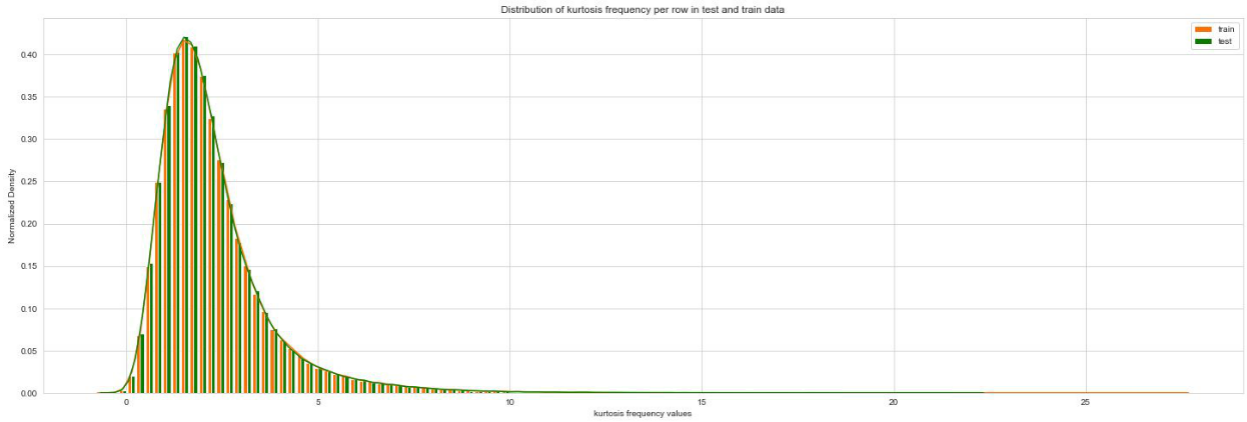


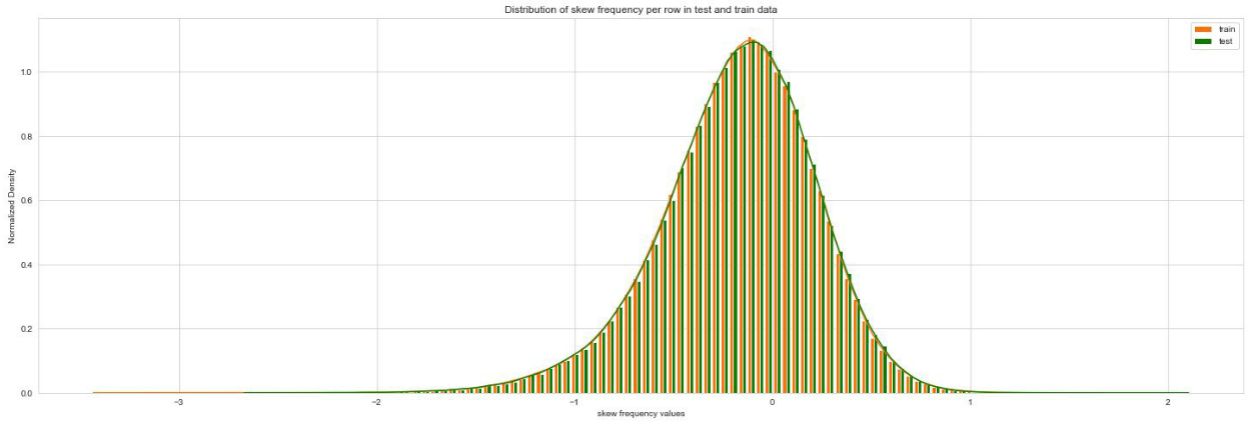
Row wise Distribution of mean, median, standard deviation, kurtosis, skewness, min, max of test and train datasets

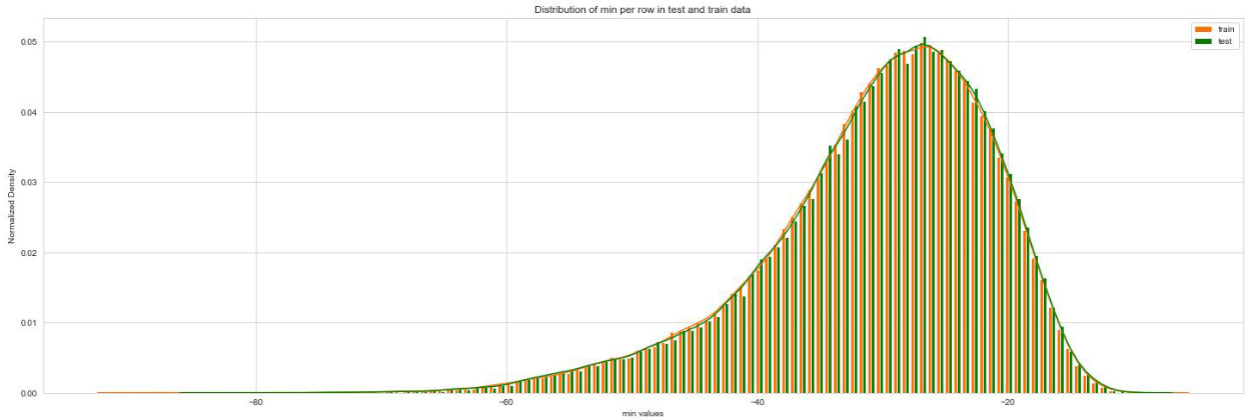


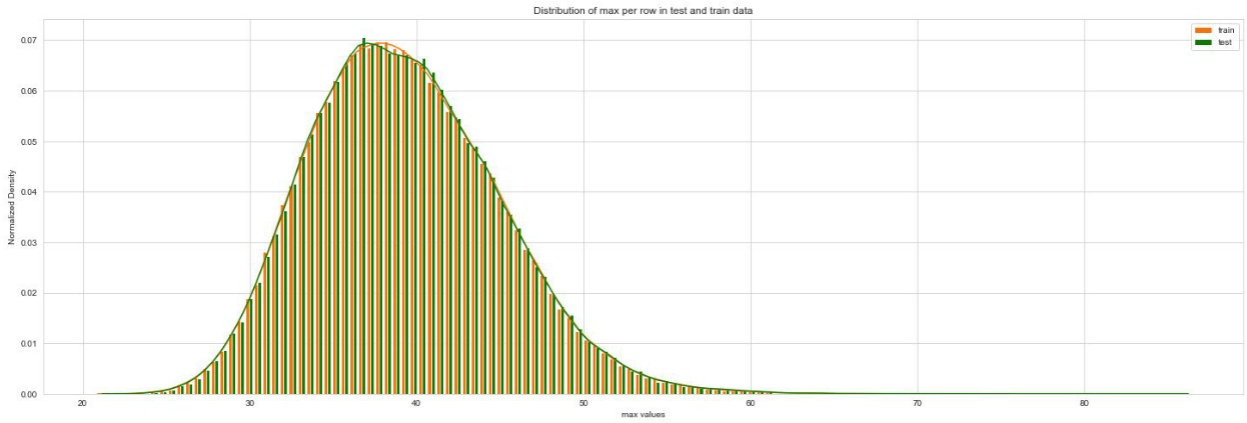

Distribution of median per row in test and train data
train
test
Normalized Density
median values


Distribution of standarad deviation per row in test and train data
train
test
Normalized Density
standarad deviation values


Distribution of kurtosis frequency per row in test and train data
train
test
Normalized Density
kurtosis frequency values







Standard deviation is relatively large for both train and test variable data min, max, mean, median, sdt values for train and test data looks quite close. Mean values are distributed over a large range. Moreover mean and median have similar distribution.
Both train and test are Leptokurtic and negatively skewed.

Outlier Analysis- I can see from the above plots, that both the training and test sets are very similar to one another and that the distributions do not have any anomalies.

Correlation among numerical variables

Minimum correlation among variables is -0.009844361358419677

Maximum correlation between variables is 0.009713658349534146

We have 200 features that are mostly uncorrelated between them.

Feature Engineering

Add descriptive statistics i.e. mean, median, standard deviation, minimum, maximum, kurtosis, skewness and sum as separate features in train dataset.

I created 200 new features based on the original 200 features where the new values for these new features would be the number of occurrences (frequency) of the original value in the training set. Often there can be relevant information in the frequency, depending on how the categories are drawn. (Using probability density of occurrence in place of frequency.)

Since the data are anonymous and I did not have information about what each of the features meant.

Dealing Imbalanced dataset

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.

Lets see these techniques-

1. Random Under-Sampling

Random Undersampling 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.

I will use Gradient boosting because it is not sensitive to outliers and noisy data (I have avoided Outlier analysis) like Adaptive Boosting.

Due to local environmental problem I do not have XGboost package. But it can be used as final algorithm due to its higher performance and speed.

I will use Recall, Precision, F1_score (harmonic mean of Precision and recall), AUC-ROC score along with Accuracy for model evaluation.

Modelling

Logistic regression

This is the classification problem. We can use logistic regression for this problem.

Logistic Regression is used when the dependent variable(target) is categorical. It has a bias towards classes which have large number of instances. It tends 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.

Naive Bayes

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. It performs well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell

curve, which is a strong assumption). Also numerical variables are very less correlated. On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.

Support Vector Machine

“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is the number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well. Support Vectors are simply the coordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line). It is effective in cases where the number of dimensions is greater than the number of samples. But it doesn’t perform well, when we have large data set because the required training time is higher.

Random Forest

Random Forest is a bagging based ensemble learning model. Random forests is slow in generating predictions because it has multiple decision trees. Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming. Thus result (AUC-score) shown below for random Forest is not modified if num round would increase the score will be better but speed will be very slow. These results are shown for default values of Parameters.

LightGBM

LightGBM is Gradient Boosting ensemble model which is faster in speed and accuracy as compared to bagging and adaptive boosting. It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST. But parameter tuning in LightGBM should be done carefully.

Following are the metrics evaluated for different classification models.

	model_name	AUC Score	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.799240	0.913317	0.264389	0.675424	0.380021
1	Random Forest	0.549762	0.899467	0.000166	0.200000	0.000331
2	Light GBM	0.844332	0.921483	0.319456	0.760063	0.449842
3	Naive Bayes	0.817300	0.919783	0.348648	0.703481	0.466230
4	SVM	0.720496	0.901417	0.205838	0.524071	0.295582

Choosing LightGBM as final model for further classification.

After applying the same lightGBM model to feature engineered data we have following results-

```
AUC Score for LightGBM is  0.8484616051852578
Accuracy Score for LightGBM is  0.9214166666666667
Precision Score for  LightGBM is  0.3114944435229723
Recall Score for LightGBM is  0.769041769041769
f1 Score for LightGBM is  0.4433951127375752
```

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.

```
AUC Score for LightGBM is  0.8854757895734049
Accuracy Score for LightGBM is  0.9178
Precision Score for  LightGBM is  0.22043456626306188
Recall Score for LightGBM is  0.8513773222293401
f1 Score for LightGBM is  0.35019762845849806
```

Further Improvements

- Further improvements in model can be done by-
- Using Parallel Processing with LightGBM Algorithm.
- Selecting important features and then modelling them.
- Using Stratified Folding for train and test splits.
- Taking a try for XGBoost for faster speeds.

Conclusion

This was a classification problem on a typically unbalanced dataset with no missing values. Predictor variables are anonymous and numeric and target variable is categorical. Visualising descriptive features and finally I got to know that these variables are not correlated among themselves. After that I decided to treat imbalanced dataset and built different models with original data and choosen LightGBM as my final model then using the same model with feature engineered data we got AUC-Score of 0.848 and after tuning parameters final value is 0.885. Then I suggested some improvements and used stratified folding for splitting between train and test data the final value of AUC-Score is 0.899.