Machine Learning Engineer Nanodegree

Capstone Project

Santander Customer Transaction Prediction

Megha Patil

March 22nd, 2019

I. Definition

Project Overview

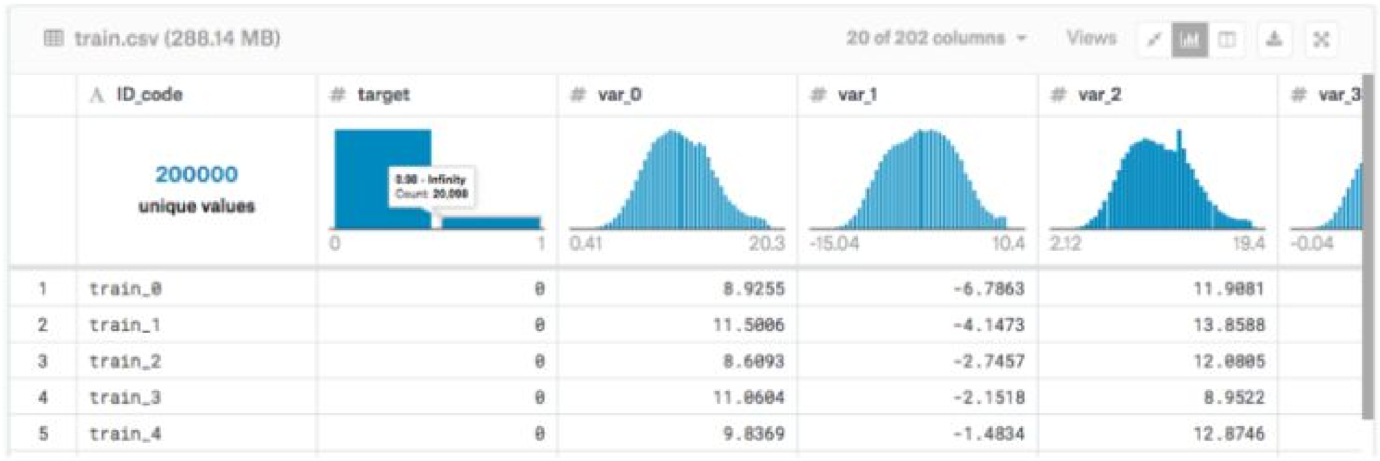
This project is based on Kaggle competition described at
https://www.kaggle.com/c/santander-customer-transaction-prediction

The problem is in finance domain. Santander is a commercial bank and financial services company. They are looking for ways to help their customers understand their financial health and identify which products and services might help them achieve their monetary goals.
The challenge is to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted. The data provided for this competition has the same structure as the real data Santander has available to solve this problem.

Problem Statement

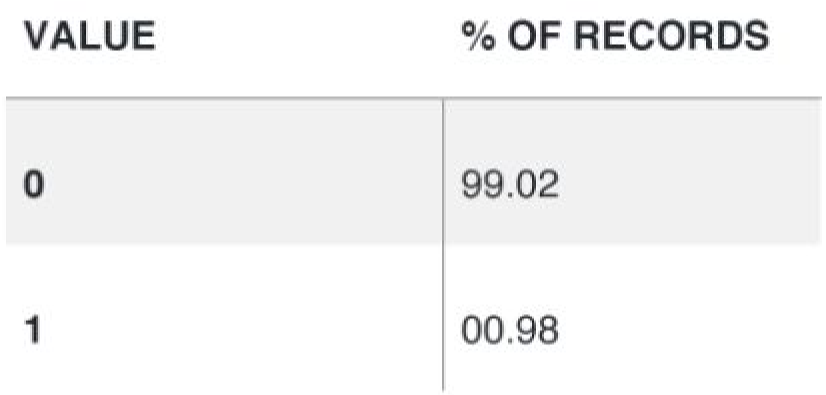
As described in challenge, we have to use various machine learning techniques on given dataset to predict which customer will make specific transaction in future. We are provided with an anonymized dataset containing numeric feature variables, the binary target column, and a string ID_code column. The task is to predict the value of target column in the test set.

Datasets and Inputs



	ID_code	target	var_0	var_1	var_2	var_3
1	train_0	0	8.9255	-6.7863	11.9081	
2	train_1	0	11.5006	-4.1473	13.8588	
3	train_2	0	8.6093	-2.7457	12.0805	
4	train_3	0	11.0604	-2.1518	8.9522	
5	train_4	0	9.8369	-1.4834	12.8746	

Santander has provided a single data file – train.csv for this problem. See above a sample of the data provided. The file is in CSV format. It contains 202 columns and 200K rows. Each row has an ID and target, followed by 200 attributes. “target” column is a binary value. Value of 1 represents that the customer will make a transaction. This is the column that needs to be predicted.
The distribution of the “target” column (column to be predicted) in the training dataset is as follows:



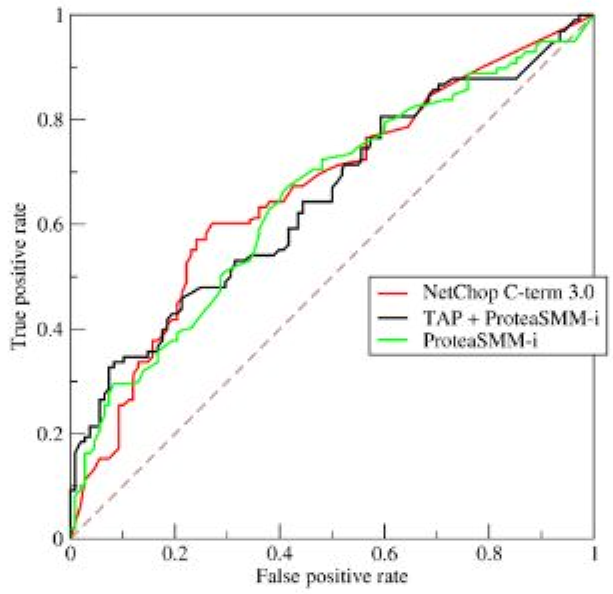
VALUE	% OF RECORDS
0	99.02
1	00.98

The value of 1 is found in less than 1% of cases. This is an **unbalanced dataset**.

1. **Imbalanced dataset** is relevant primarily in the context of supervised machine learning involving two or more classes.
2. **Imbalance** means that the number of data points available for different the classes is different.

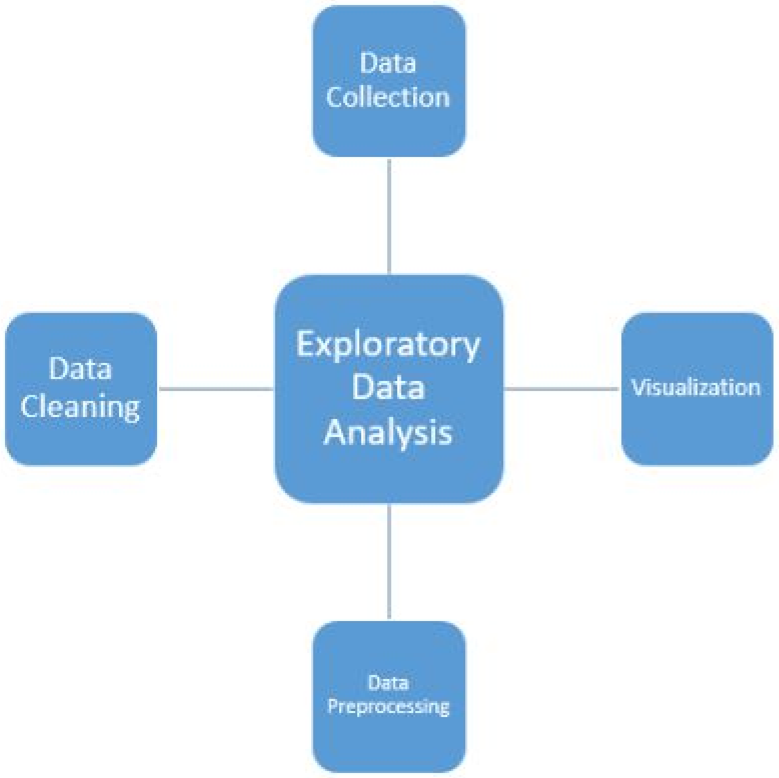
Metrics

The solution is expected to predict which customer will make a transaction. The prediction will be based on the 200 attributes provided in Santander data. The prediction will be specified by a binary variable against the ID specified in the input. Value of 1 indicates a prediction that the customer will make a transaction in future.
Solution will be evaluated on area under the ROC curve between the predicted probability and the observed target.

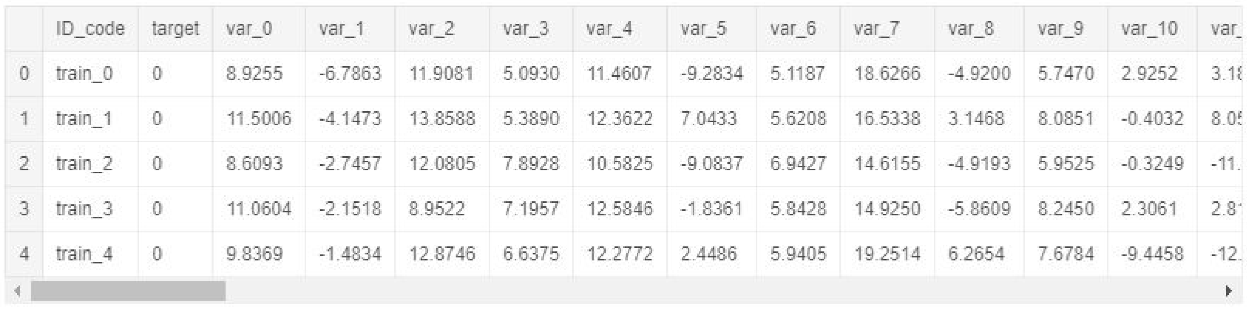


II. Analysis

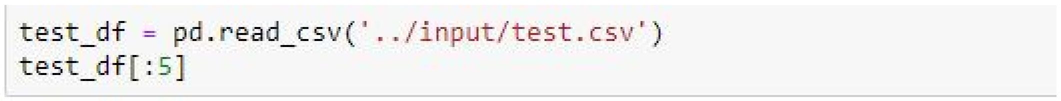
Data Exploration



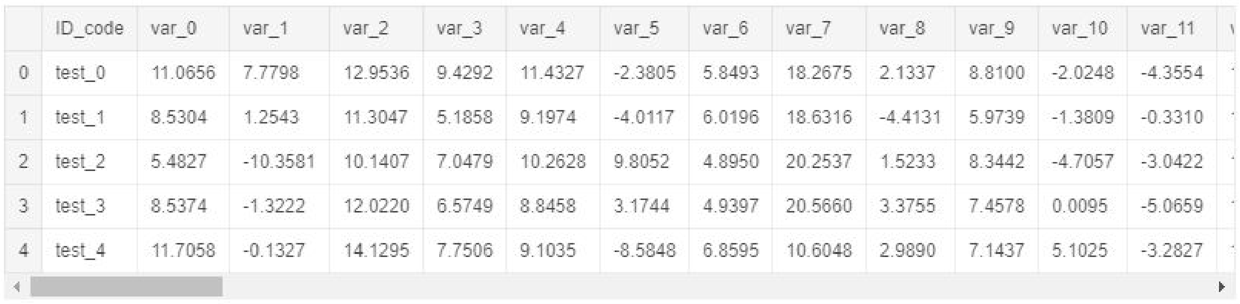
As stated earlier we are provided with anonymized dataset containing numeric feature variables, the binary target column, and a string ID_code column.The task is to predict the value of target column in the test set.



	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.7470	2.9252	3.18
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	3.1468	8.0851	-0.4032	8.05
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.9525	-0.3249	-11.
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.2450	2.3061	2.81
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.6784	-9.4458	-12.



```
test_df = pd.read_csv('../input/test.csv')
test_df[:5]
```



	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11
0	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	8.8100	-2.0248	-4.3554
1	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	5.9739	-1.3809	-0.3310
2	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	8.3442	-4.7057	-3.0422
3	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	7.4578	0.0095	-5.0659
4	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	7.1437	5.1025	-3.2827

Train contains:

- **ID_code** (string);
- **target**;
- **200** numerical variables, named from **var_0** to **var_199**;

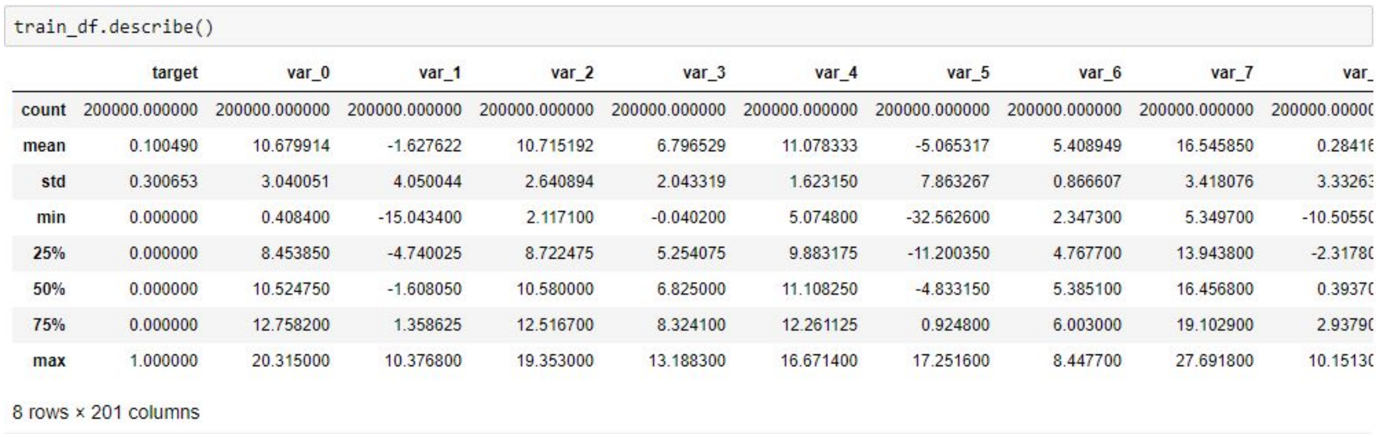
Test contains:

- **ID_code** (string);
- **200** numerical variables, named from **var_0** to **var_199**;

Each row has an ID and target, followed by 200 attributes. "target" column is a binary value. Value of 1 represents that the customer will make a transaction. This is the column that needs to be predicted.

As shown in distribution of target column in training dataset, the dataset is unbalanced.

There are no missing data in train and test datasets.



```
train_df.describe()
```

	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.00000
mean	0.100490	10.679914	-1.627622	10.715192	6.796529	11.078333	-5.065317	5.408949	16.545850	0.2841
std	0.300653	3.040051	4.050044	2.640894	2.043319	1.623150	7.863267	0.866607	3.418076	3.3326
min	0.000000	0.408400	-15.043400	2.117100	-0.040200	5.074800	-32.562600	2.347300	5.349700	-10.5055
25%	0.000000	8.453850	-4.740025	8.722475	5.254075	9.883175	-11.200350	4.767700	13.943800	-2.3178
50%	0.000000	10.524750	-1.608050	10.580000	6.825000	11.108250	-4.833150	5.385100	16.456800	0.3937
75%	0.000000	12.758200	1.358625	12.516700	8.324100	12.261125	0.924800	6.003000	19.102900	2.9379
max	1.000000	20.315000	10.376800	19.353000	13.188300	16.671400	17.251600	8.447700	27.691800	10.1513

8 rows × 201 columns

We can make few observations here:

- standard deviation is relatively large for both train and test variable data;
- min, max, mean, sdt values for train and test data looks quite close;
- mean values are distributed over a large range.

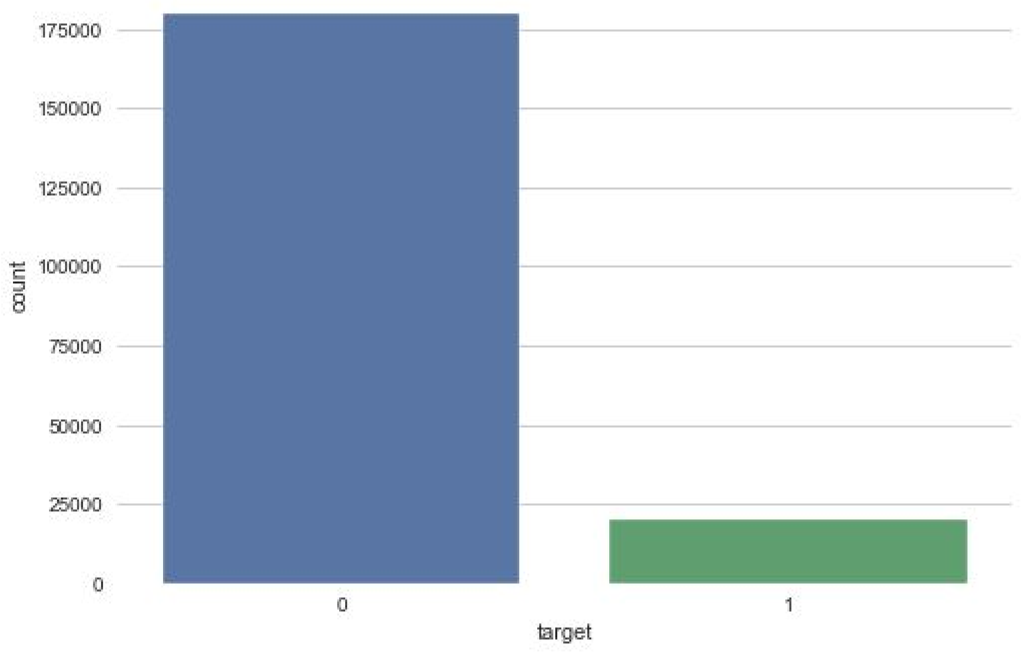
The number of values in train and test set is the same.

Exploratory Visualization

Let's check the distribution of **target** value in train dataset.

```
target = train_df['target']
train = train_df.drop(["ID_code", "target"], axis=1)
sns.set_style('whitegrid')
sns.countplot(target)
```

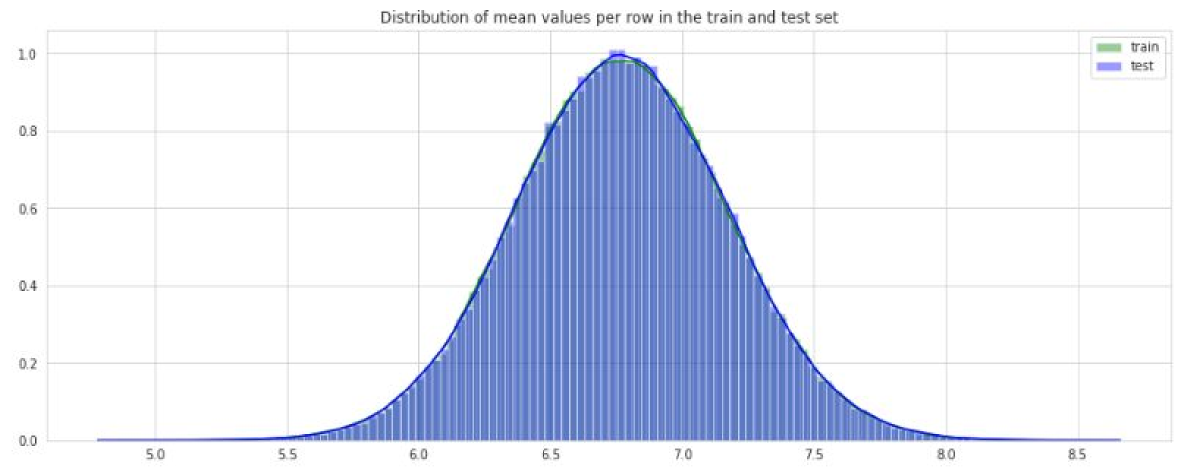
```
<matplotlib.axes._subplots.AxesSubplot at 0x19382c74cc0>
```



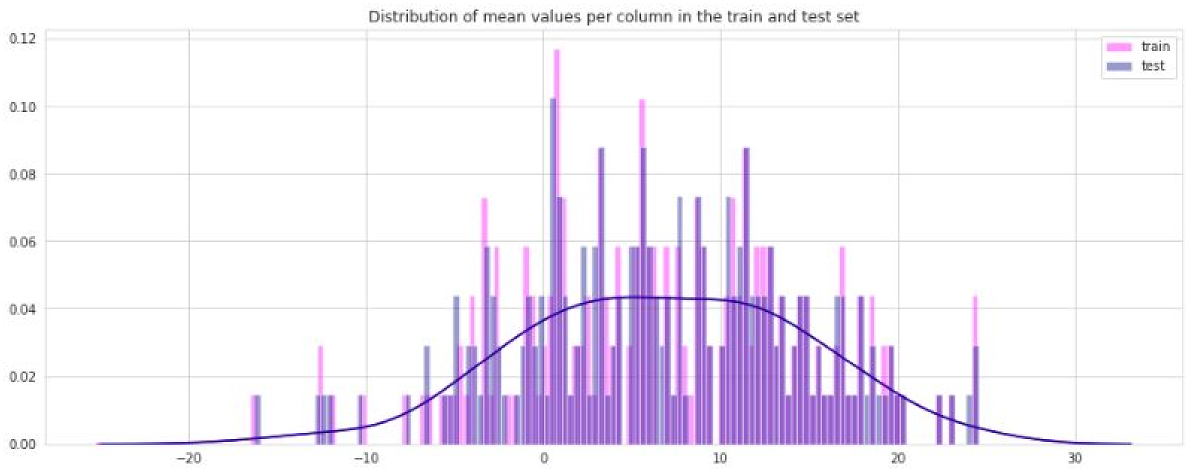
There are 10.049% target values with 1 The data is unbalanced with respect with target value.

Distribution of mean and std

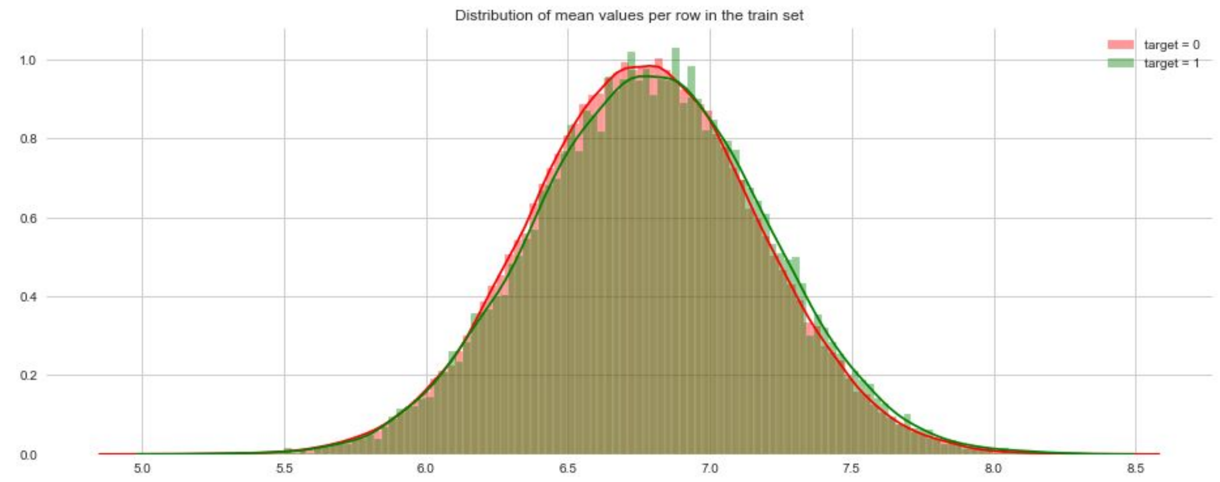
Let's check the distribution of the mean values per row in the train and test set.



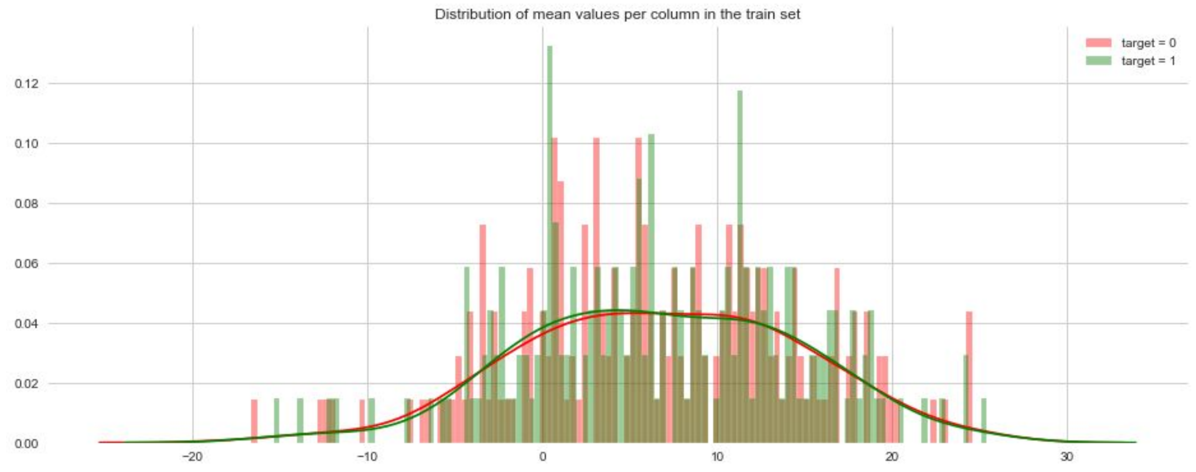
Let's check the distribution of the mean values per columns in the train and test set.



Let's show the distribution of standard deviation of values per row for train and test datasets.

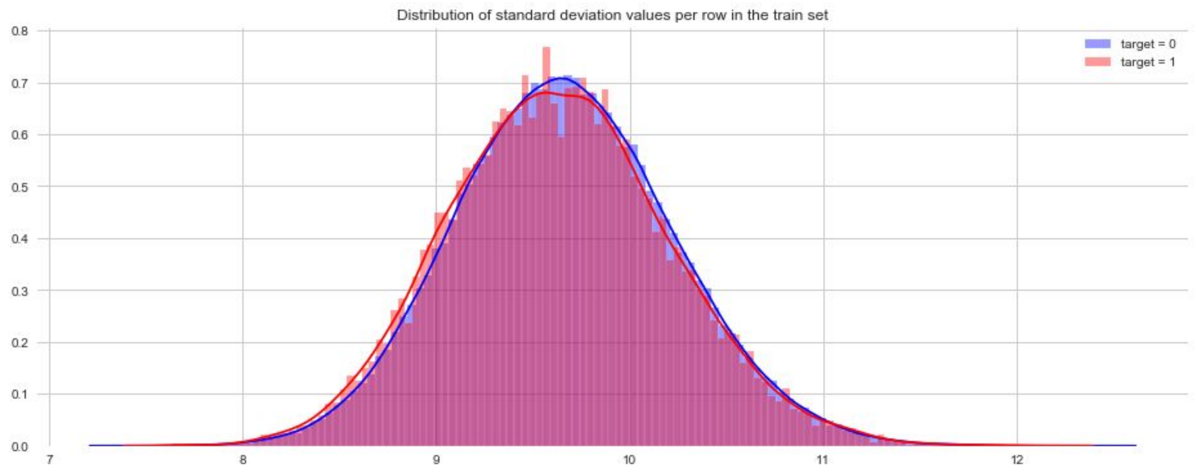


Let's check the distribution of the standard deviation of values per columns in the train and test datasets.

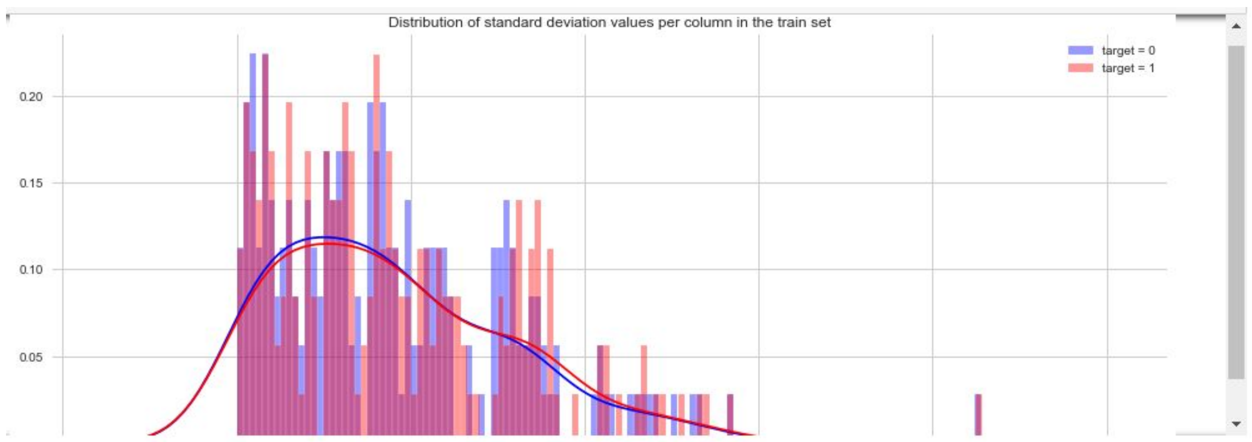


Code

The distribution of the standard deviation per row in the train dataset, grouped by value of target:



Let's check the distribution of standard deviation per columns in the train and test datasets.



Algorithms ,Techniques

For this problem we will try to implement following models:

1. Linear Regression

   We start with most basic algorithm used for classification problems. Initial model with defining only the regularization parameter (C) yielded 0.6 AUC. Since this is an unbalanced dataset, we need to define another parameter 'class_weight = balanced' which will give equal weights to both the targets irrespective of their representation in the training dataset.

2. Random Forest

   Built random forest model with parameters like class_weight, random_state, and hyperparameters like max_features and min_sample_leaf as earlier. We have also defined the n_estimators which is a compulsory parameter. This defines the number of decision trees that will be present in the forest.

3. Decision Trees:
   Moving on to a slightly advanced algorithm, decision trees. Again, the parameters here are class_weight to deal with unbalanced target variable, random_state for reproducibility of same trees. The feature max_features and min_sample_leaf are used to prune the tree and avoid overfitting to the training data.

   Max_features defines what proportion of available input features will be used to create tree.

   Min_sample_leaf restricts the minimum number of samples in a leaf node, making sure none of the leaf nodes has less than 80 samples in it. If leaf nodes have less samples it implies we have grown the tree too much and trying to predict each sample very precisely, thus leading to overfitting.

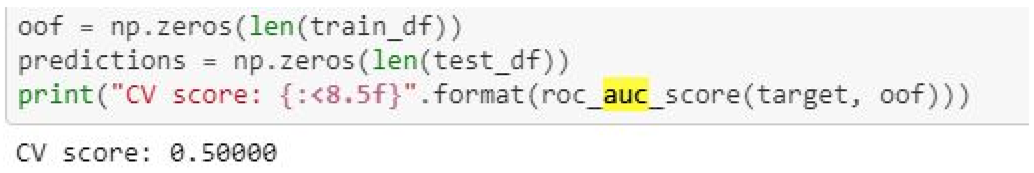
4. Light Gradient Boosting Method

   Light GBM is a gradient boosting framework that uses tree based learning algorithm. It grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

   It is 'Light' because of its high speed. It can handle large data, requires low memory to run and focuses on accuracy of results. Also supports GPU learning and thus data scientists/ Kagglers are widely using LGBM for data science application development.

Benchmark

I implemented a naïve solution as benchmark model. The training data distribution shows that "target" has value 1 in less than 1% of records. In other words, the transaction is expected to happen in very rare cases. We will implement a naïve model which always predicts "target" value of 0. This will be our model to use as benchmark. We will use the evaluation metric - "**Area under the ROC curve between the predicted probability and the observed target**" - to measure the performance of benchmark model. The solution we develop must do better than the benchmark model in predicting transactions.



```
oof = np.zeros(len(train_df))
predictions = np.zeros(len(test_df))
print("CV score: {:<8.5f}".format(roc_auc_score(target, oof)))

CV score: 0.50000
```

Here benchmark model have a score of **0.5** on the chosen metric. Our implementation must beat this benchmark.

III. Methodology

Data Preprocessing

For this project, the data has been provided by Santander bank. I verified that the data is in a format that is appropriate for machine learning. I verified the following attributes of the data.

All the feature columns are specified as numerical values. There are no string labels or categorical values. Hence, data transformation - like on-hot encoding is not required.

The values are populated completely. Hence we do not have to handle null/blank cases.

The distribution of values in the train and test data is similar. I have plotted mean, standard deviation, distribution of the columns to check the distribution of data.

Implementation

I have used the following main main technologies in the project:

1. Jupyter Notebook as IDE
2. Python3 as programming language
3. Numpy and Pandas modules for numeric functions, algorithms and data manipulation
4. Scikit-learn for Machine Learning models.Lightgbm module for Light Gradient Boosting model.
5. Matplotlib for plotting graphs.

I solved this problem by implementing the following machine learning models:

1. Linear Regression
2. Decision Trees
3. Random Forest
4. Light Gradient Boosting Model

For each model implementation I followed the following steps:

1. Split the training dataset provided by Santander into train and test data set.
2. Create the model.
3. Train the model using train part data set.
4. Validate the model using test part of data set.
5. I used Stratified K-folds technique. This is a good for cross-validation.
6. Measure performance using “Area under ROC curve” method.
7. Run the model on the testing data set provided by Santander.
8. Submit the predictions on test data to Kaggle and get the AU-ROC score on testing data.

Refinement

After trying out 4 different models, I decided to go with Gradient Boosting Decision Tree model. I further experimented with various model parameters to find the optimal parameters where score was maximum and pace of training was acceptable.

IV. Results

Model Evaluation and Validation

I found the best performance with Lightgbm model. So I further refined the model using following parameters. I have listed below the final values of the parameters:

1. bagging_fraction = 0.38. This causes the model to randomly select part of data without resampling.
2. bagging_freq = 5: frequency for bagging. Model performs bagging after every 5 iterations.
3. boosting = ‘gbdt’ (Gradient Boosting Decision Tree)

4. feature_fraction = 0.04. LightGBM will randomly select part of features on each iteration. This fraction helps speed up training and helps avoid over-fitting
5. min_data_in_leaf = 80. This helps in avoiding overfitting by making sure that decision tree does not branch too much to fit all data points.
6. reg_alpha, reg_lambda : L1 and L2 regularization parameters respectively.

The final model is very successful in predicting the customer transactions. The score on “Area under the ROC curve” metric is around 0.9. This indicates the model can fairly accurately predict the target value.

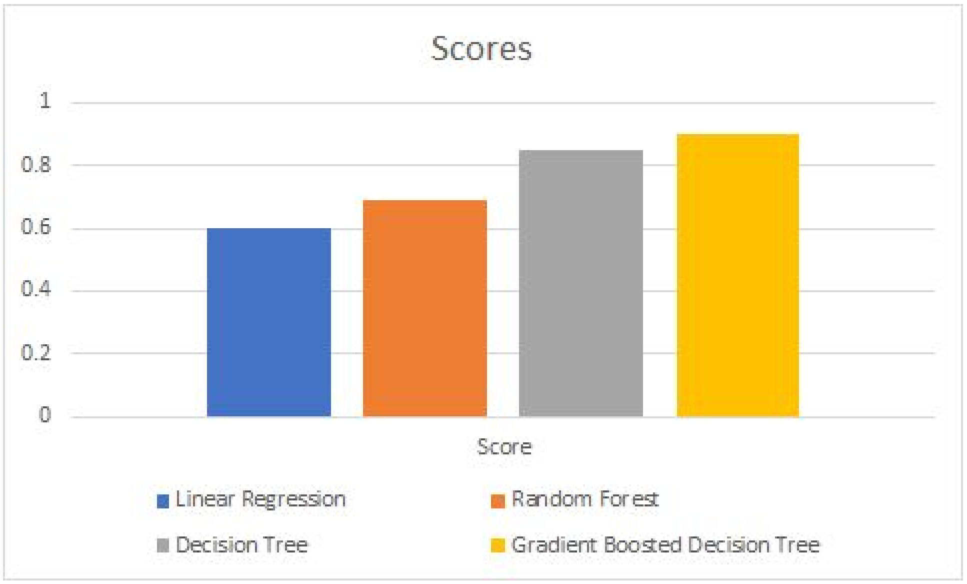
Justification

The benchmark model scored around 0.5 for the selected metric (Area under ROC curve).

The final model (Gradient Boosting Decision Tree) scored around 0.9 on this metric. This indicates that I was successful in achieving the goal of creating and training a model using the given training data, that can accurately predict customer transactions.

V. Conclusion

Free-Form Visualization



Reflection

I have been able to successfully predict customer transactions based on given data attributes. This is a good outcome for the project. This establishes that the machine learning techniques used here have practical application in the field of finance. This can be useful in predicting if a customer will buy a certain product or to predict loan default, fraud or other such anomalous transaction.

The Gradient Boosting Decision Trees model gave excellent performance for this problem. LightGBM library I used gives good performance and is easy to use.

Improvement

Improvement can be done in two ways:

1. The score of 0.90 can be further improved.
2. I have not made a difference between past and future transactions. This code should be further tested to check if past inputs can help in predicting future transactions.