MSBA 6420
Predictive Analytics

Ankur Aggarwal, Anurag Mishra
Sheetal Pasam, Sourabh Koul, Sricharan Sridhar

UNIVERSITY OF MINNESOTA
Driven to Discover℠

Santander Customer Transaction Prediction

Final Project - Spring 2022

Table of Contents

Definition and Introduction

Santander is 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.

We come in to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

Data Source & Overview

We are provided with an anonymized dataset containing numeric feature variables, the binary 'target' column, and a string 'ID_code' column.

The training dataset consists of 200 numeric variables labelled from var_0 to var_199 which will be used to train the model

Task

Predict the value of the 'target' column in the test set, which is the probability that the user is making a purchase or not.

Technical Specifications

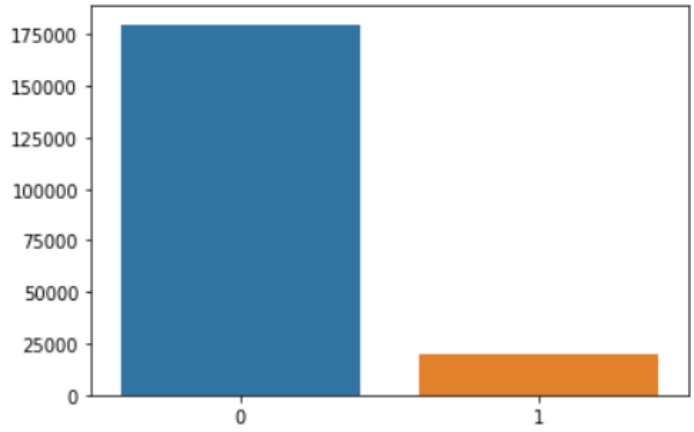
<table>
<tr><th>Packages</th><th>Environments</th><th>Operating Systems</th></tr>
<tr><td>keras</td><td rowspan="2">Google Colab</td><td rowspan="2">MacOS</td></tr>
<tr><td>pandas</td></tr>
<tr><td>matplotlib</td><td rowspan="2">Jupyter Notebook</td><td rowspan="4">Windows 11</td></tr>
<tr><td>sklearn</td></tr>
<tr><td>tensorflow</td><td rowspan="2">Notepad++</td></tr>
<tr><td>seaborn</td></tr>
</table>

Exploratory Data Analysis

To understand the raw data better, we had to analyze distributions to check for any inherent patterns. Since our dataset had 200 numeric variables, we chose a couple of variables to identify their distribution

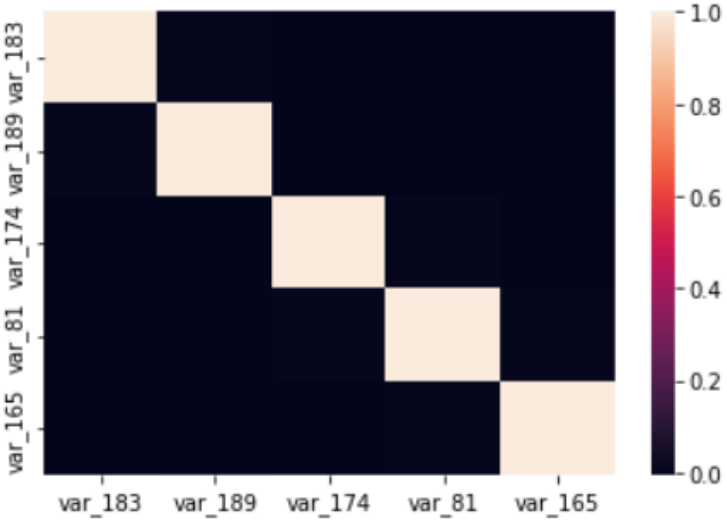
Class Imbalance

Right off the bat, we notice that we have highly skewed data concerning the target label. There were ~9x more data points for people who did not purchase than for people who did.



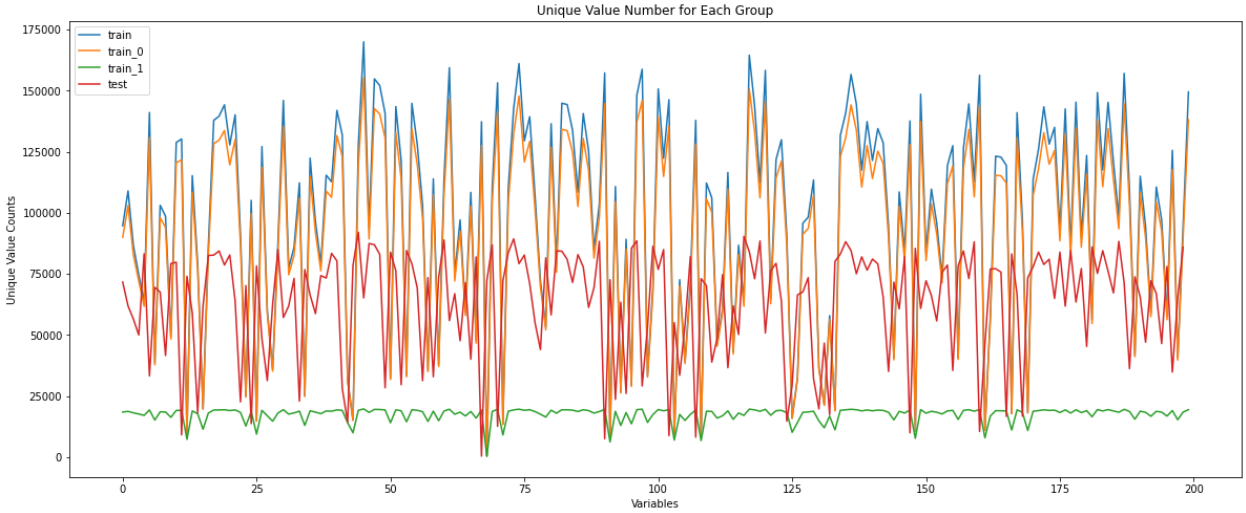
Correlation among variables

We plotted a correlation matrix and found that the correlations among the features are very low. The below heatmap shows the top 5 most correlated features.



Synthetic Data

When the number of unique values for each train and test dataset (split by positive and negative labels) was plotted, this was the output:



There is a significant difference in the unique value numbers of each variable, between training data and test data.

Some noises in the original training data must've been added by Kaggle hosts. So if the variable value is unique, this variable may be synthetic (added noise from the original data), so we need to tell this information to the model.

Now we know that noise and synthetic data exist in both training and testing datasets. To tackle this problem, we added 600 new features which included statistics of the pre-existing 200 columns of data, to capture the authenticity of the data point. Due to the high volume of data, some models use a sample of the true data (~116k records)

Data Scaling

Each feature variable seems to have numeric values from different ranges. To make sure no variable is overwhelmingly affecting the model, we need to transform each feature so that its values are brought to a standard comparable range.

Due to the existence of outliers, the min-max scaler would be ineffective. The outliers would pull the extreme values making the 'normal' values obsolete. Hence, we decided to use StandardScaler.

Model Training

Individual models were trained using the transformed data. To find the best parameters for the models, we used nested K-Fold cross-validation and BayesSearchCV. Once the best parameters were computed, these models were stacked to create an ensemble model.

Individual Models

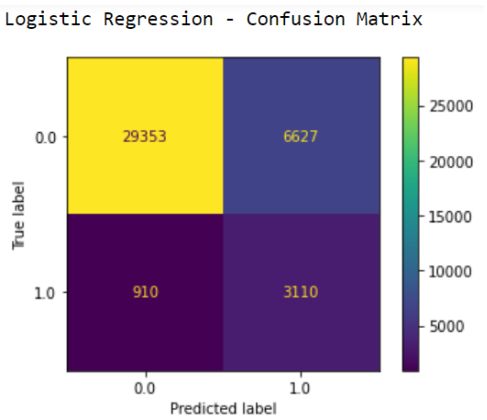
Logistic Regression

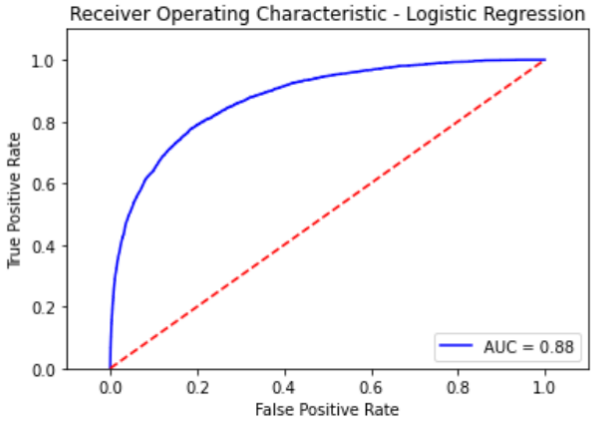
Logistic Regression is a form of classification technique used to perform binary or multi-class classification based on a set of independent variables. In binary classification, it uses probability thresholds to predict the class as 0 or 1. To balance the imbalanced classes in our data, we passed 'class_weights' as balanced while training the model.

We got the following best hyperparameters after running GridSearchCV:

C: 0.1,
class_weigh': 'balanced',
penalty: 'l1',
solver: 'saga'

	precision	recall	f1-score	support
0.0	0.97	0.82	0.89	35980
1.0	0.32	0.77	0.45	4020
accuracy			0.81	40000
macro avg	0.64	0.79	0.67	40000
weighted avg	0.90	0.81	0.84	40000

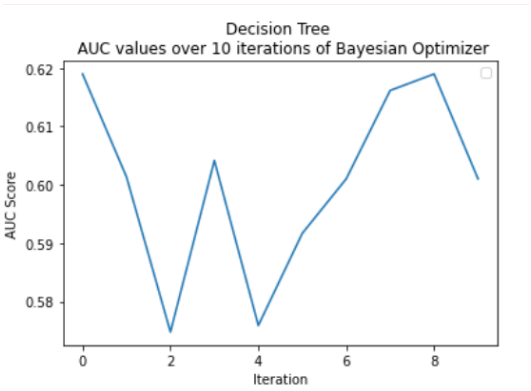


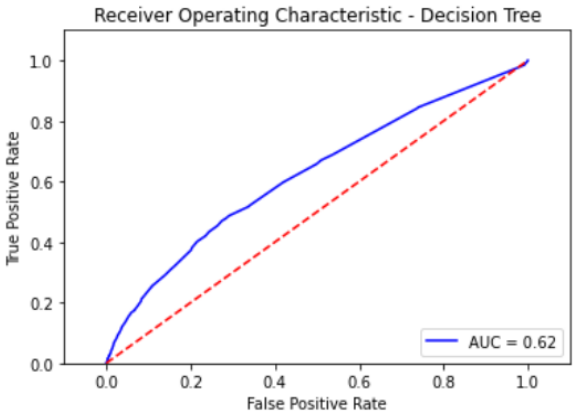


Decision Tree

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

We performed tuning on hyperparameters like max_depth, max_features and min_sample_split



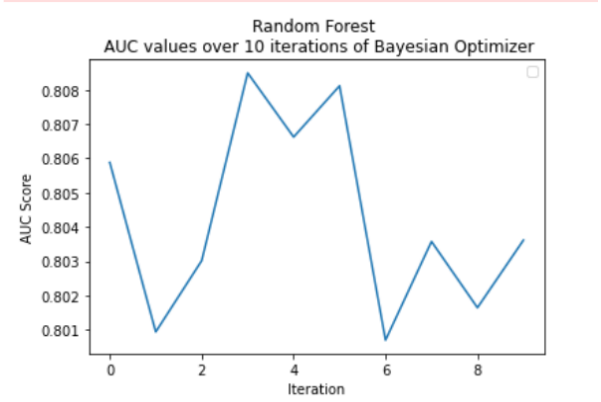


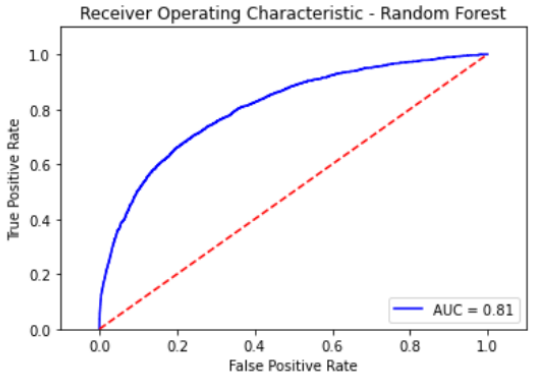
Random Forest

The random forest is a classification algorithm consisting of many decision trees. It tries to create an uncorrelated forest of trees through a bagging process and its overall prediction is more accurate than that of any individual tree.

We performed tuning on hyperparameters like:

max_depth, max_features & min_sample_split

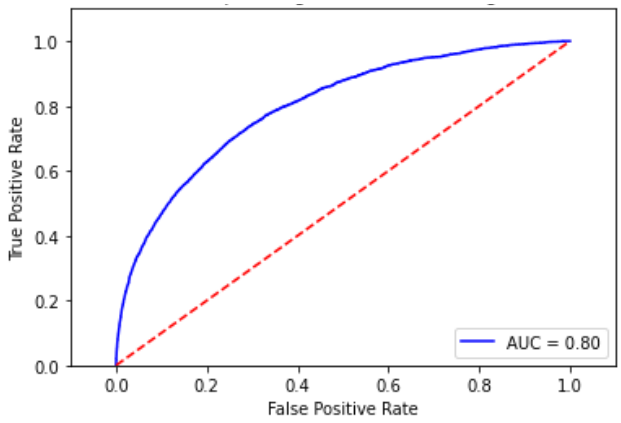




Neural Network

Neural networks try to mimic the human brain and solve complex problems by recognizing hidden patterns in data. For our dataset, we have built a 1D Convolutional Neural Network (CNN) with 5 hidden layers having 32, 24, 20, 16, and 4 neurons respectively. We used batch normalization to normalize the hidden layers and used 'elu' as our activation function. We also used Average pooling to create downsampled features as input to the next layer.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv1d_5 (Conv1D)            (None, 160, 32)           192

 batch_normalization_4 (Batc  (None, 160, 32)           128
 hNormalization)

 conv1d_6 (Conv1D)            (None, 160, 24)           792

 batch_normalization_5 (Batc  (None, 160, 24)           96
 hNormalization)

 conv1d_7 (Conv1D)            (None, 160, 20)           500

 batch_normalization_6 (Batc  (None, 160, 20)           80
 hNormalization)

 conv1d_8 (Conv1D)            (None, 160, 16)           336

 batch_normalization_7 (Batc  (None, 160, 16)           64
 hNormalization)

 conv1d_9 (Conv1D)            (None, 160, 4)            68

 average_pooling1d_1 (Averag  (None, 80, 4)             0
 ePooling1D)

 flatten (Flatten)            (None, 320)               0

 dense_1 (Dense)              (None, 1)                 321

=================================================================
Total params: 2,577
Trainable params: 2,393
Non-trainable params: 184
_________________________________________________________________
```



Ensemble Models

Boosting

XGBoost

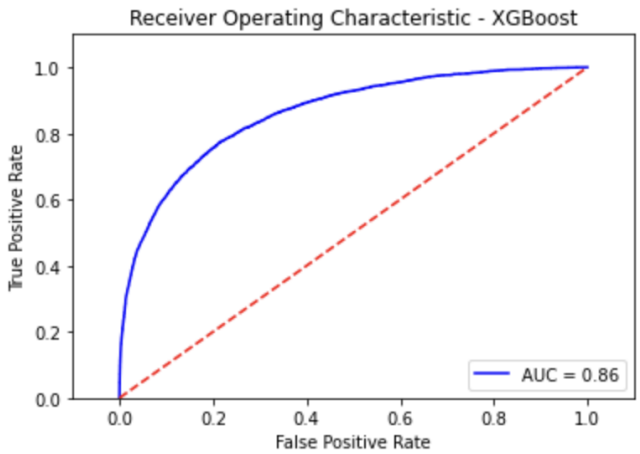
XGBoost is an optimised gradient boosting library that has been designed for efficiency, flexibility, and portability. It uses the Gradient Boosting framework to create machine learning algorithms. XGBoost is a parallel tree boosting algorithm that solves a variety of data science issues quickly and accurately. To deal with class imbalance in the data, we used.

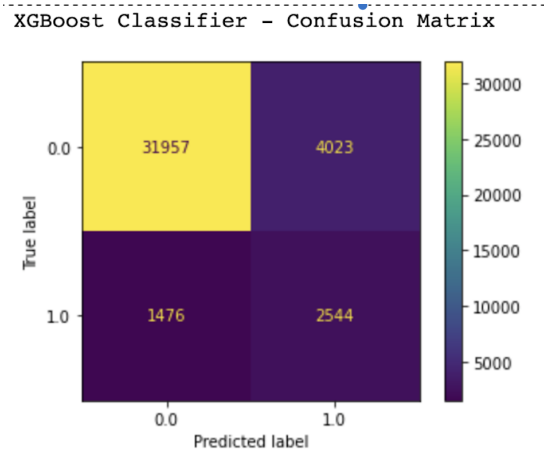
$$scale\ pos\ weight\ = \frac{count\ of\ negative\ class}{count\ of\ positive\ class}$$

$$best\ params\ = \{\text{'n\_estimators': 200,}\ \text{'max\_depth': 5,}\ \text{'learning \_rate': 0.05}\}$$

Output:

	precision	recall	f1-score	support
0.0	0.96	0.89	0.92	35980
1.0	0.39	0.63	0.48	4020
accuracy			0.86	40000
macro avg	0.67	0.76	0.70	40000
weighted avg	0.90	0.86	0.88	40000



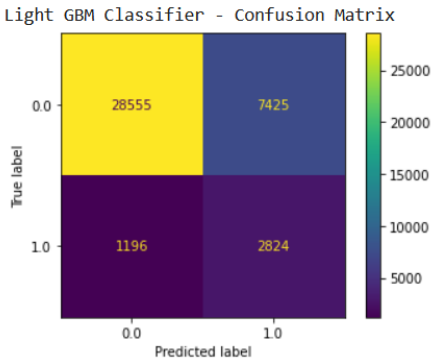


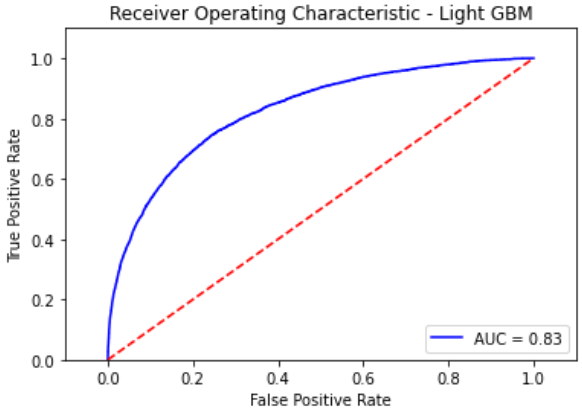
Light GBM

Light GBM is a fast and high-performance gradient boosting algorithm based on decision trees to reduce memory usage and increase efficiency. It splits the tree leaf wise whereas other boosting algorithms split the tree level-wise or depth-wise.

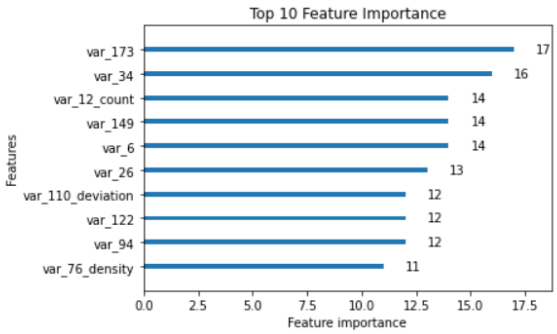
Best Params:

bagging_fraction=0.331, bagging_freq=5, boost='gbdt',
boost_from_average='false', class_weight='balanced',
feature_fraction=0.0405, learning_rate=0.0083, metric='auc',
min_data_in_leaf=80, min_sum_hessian_in_leaf=10.0, num_leaves=13,
num_threads=8, objective='binary', tree_learner='serial', verbosity=1





	precision	recall	f1-score	support
0.0	0.96	0.79	0.87	35980
1.0	0.28	0.70	0.40	4020
accuracy			0.78	40000
macro avg	0.62	0.75	0.63	40000
weighted avg	0.89	0.78	0.82	40000



Stacking [best model]

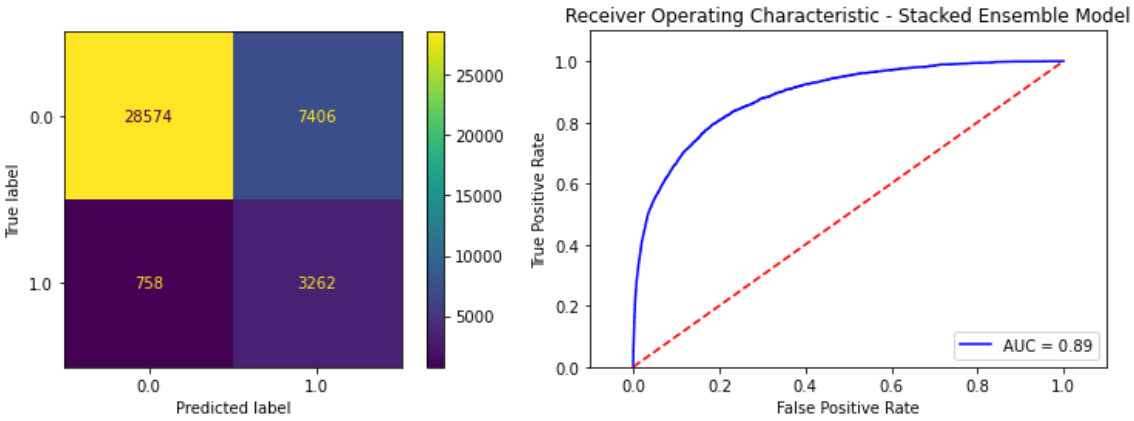
The models and their best parameters found in the individual models above were compared and analyzed to select 3 models to build a stacked ensemble model. The following configuration was used:

Base Estimators : [LightGBM, Logistic Regression]
Final Estimator : [XGBoost Classifier]

Estimators

```
In [8]:
def get_stacked_model():
    # Logistic Regression
    lr_stack = linear_model.LogisticRegression(C=0.1,
                                               class_weight='balanced',
                                               penalty='l1',
                                               solver= 'saga',
                                               random_state=RAND_STATE,
                                               verbose = 1)

    # XGBoost Classifier
    xgb_stack = XGBClassifier(max_depth=5,
                              n_estimators=200,
                              random_state=RAND_STATE,
                              verbosity = 1,
                              scale_pos_weight = 8,
                              learning_rate = .05)

    lgbm_stack = LGBMClassifier(bagging_fraction=0.331, bagging_freq=5,
                                boost='gbdt', boost_from_average='false',
                                class_weight='balanced', feature_fraction=0.045,
                                learning_rate=0.0083, metric='auc',
                                min_data_in_leaf=80,
                                min_sum_hessian_in_leaf=10.0, num_leaves=13,
                                num_threads=8, objective='binary',
                                random_state=RAND_STATE,
                                tree_learner='serial', verbosity=1)

    # Initializing the base layer of models
    base_estimators = [('xgb', lgbm_stack),
                       ('lr', lr_stack)]

    # Initiating the stacking classifier using xgboost as the final estimator
    model = StackingClassifier(estimators=base_estimators,
                               final_estimator=xgb_stack,
                               passthrough = True,
                               verbose = 1,
                               n_jobs = 3)
    return model
```

Validation Results



Test Result on Kaggle competition

Leaderboard

Raw Data | Refresh

YOUR RECENT SUBMISSION

kaggle9.csv
Submitted by Sricharan Sridhar · Submitted just now

Score: 0.88881
Public score: 0.89157

Jump to your leaderboard position

Analysis

The models are compared based on a base performance metric - Area under the Curve.

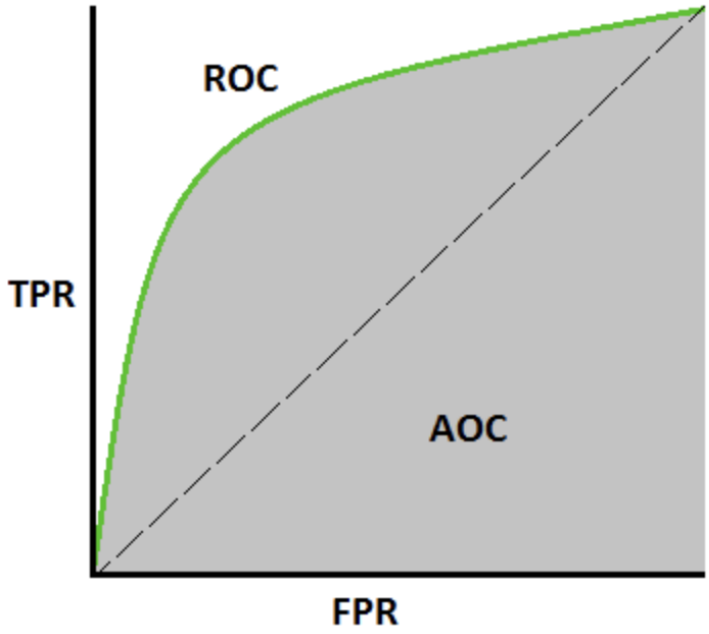
Methodology - AuC

The AuC-RoC curve is a great performance measure for classification problems.

When we need to check or visualize the performance of the multi-class classification problem, we use the AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. It is also written as AUROC (Area Under the Receiver Operating Characteristics).

The ROC curve is plotted with the True Positive Rate (TPR) against the False Positive Rate (FPR) where TPR is on the y-axis and FPR is on the x-axis.

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}$$

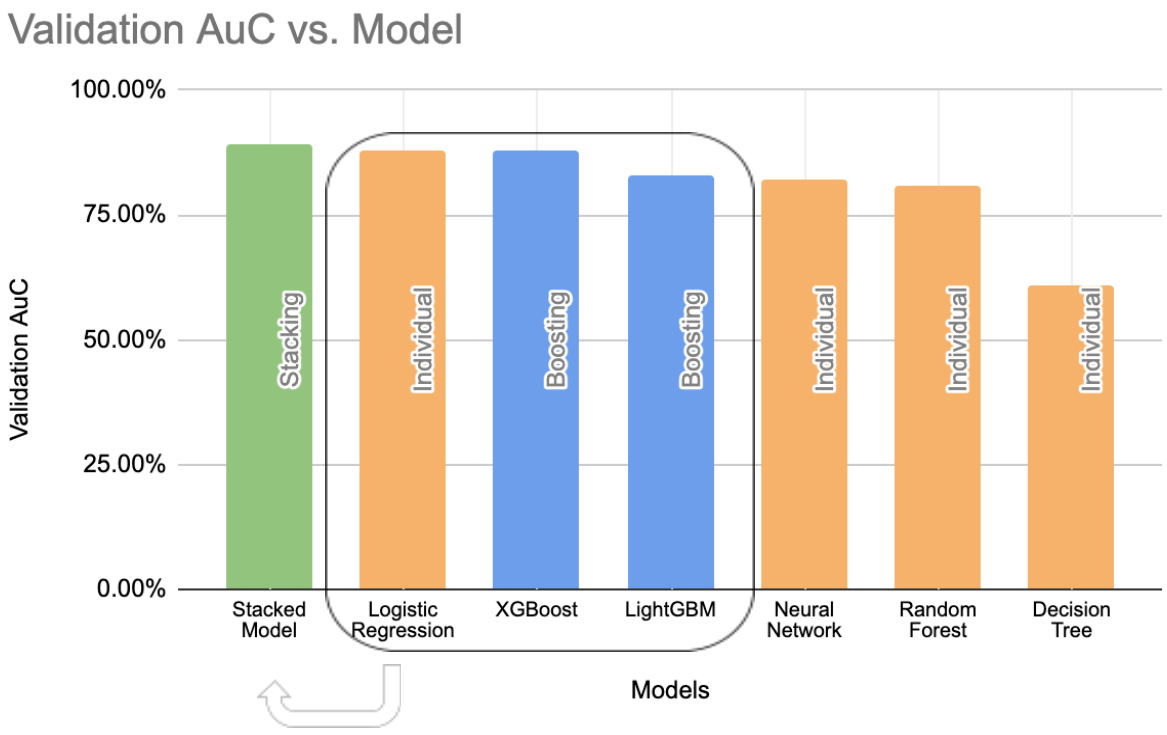


Model Comparison

The AuC scores of all models were collated and compared to see which model is giving us the best performance. In our scenario, we find out that Logistic Regression, XGBoost Classifier and LightGBM are our top 3 performing models.

We proceeded to choose these three models to build our stacked ensemble model. The ensemble model built resulted in a slightly better performance than the individual models. Overall, stacking the models increased our AuC by ~1%.

Model	Validation AuC	Type
Stacked Model	89.00%	Stacking
Logistic Regression	88.00%	Individual
XGBoost	88.00%	Boosting
LightGBM	83.00%	Boosting
Neural Network	82.00%	Individual
Random Forest	81.00%	Individual
Decision Tree	61.00%	Individual



Summary

EDA: Data was explored and the presence of noise and synthetic data was noticed. Added 600 new variables on top of the existing 200 to address the synthetic data issue.

Model Comparison: All individual models were compared based on the AuC under RoC. The 3 best performing models on validation data were used to create a stacking ensemble model.

Final model used for classification: Stacking ensemble models using LightGBM and logistic regression models as base estimators, and XGBoost model as the final estimator

AUC obtained using the final model: 0.89

Cost Analysis

Source of reference for cost benchmarks: https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)

Taking the benchmarks from a German-based banking data and using those benchmarks in our model, we assume that the cost of misclassifying a customer who would have actually made a transaction is 5 times the cost of misclassifying a customer who would not have made the transaction.

Based on the confusion matrix we obtained above, our misclassification cost is:

758*5 + 7,406*1 = **$11,196**

References

- https://www.kaggle.com/code/nawidsayed/lightgbm-and-cnn-3rd-place-solution/notebook
- https://www.kaggle.com/code/mks2192/list-of-fake-samples-and-public-private-lb-split/notebook
- https://github.com/tianqwang/Santander-Customer-Transaction-Prediction