Machine Learning Engineer Nanodegree

Capstone Project

Tareq Mufaddi
September, 2018

I. Definition

Project Overview

Customer satisfaction plays a pivotal role in any successful business, it has been and always will be one of the critical success factors monitored closely by the management. Identifying dissatisfied customers early would allow for proactive actions to take place before customers leave to the competition [1]. It has been observed that the cost of acquiring a new customer could reach five to six times higher than maintaining an existing customer [2], and studies have shown that customer retention of only 5% can increase profits by almost 100% [4]. At Santander, customers are in the center of the business model and the need for better innovative tools to better understand customer needs is key for a long-lasting relationship [5]. For this reason, Santander Bank has created a $200 million fintech fund to invest in startups linked to artificial intelligence, and the bank believes that the role of artificial intelligence goes beyond automation and NLP to reach cost reduction and improving efficiency [6].

This area presents itself as a great opportunity for the implementation of machine learning for several reasons. Firstly, the availability of customer data is not something new for companies in general, this means such data can be readily available and does not require major investments and preparation time. The data itself might be of low quality, but various tools have been developed to deal with these limitations via data munging and wrangling. Also, given the well-established link between customer satisfaction and business performance, decision makers will be more easily convinced to implement new state-of-the-art technologies in this area which then could open the door for machine learning to take part in other areas related to business performance thought not applicable before.

For this capstone project, I plan to explore which model would best fit this classification problem based on the knowledge I have gained throughout this Nanodegree and the customer dataset provided by Santander. I am personally interested in the area of business performance and found this to be a very good starting point.

Problem Statement

Santander has created a Kaggle competition to help them retain unsatisfied customers. The first step is to be able to identify them before leaving, and this is the problem. By providing a customer dataset, Santander needs an algorithm which can **predict the probability** of the customer being **satisfied** or **unsatisfied** based on hundreds of anonymized customer related features. This will give the bank enough time to act before it's too late [7].

The general approach will be to first have a look on the dimensionality of the dataset and data types to check which transformations are required to prepare the dataset for further analysis. The number of dimensions and the sample size can give information to which algorithms can better fit the data. Next, the evaluation metric will be selected, and a linear benchmark model will be tested to get an intuition about the linearity of the solution. In the case of larger datasets, dimensionality reduction will play an important role in data preprocessing, this can be done by either employing an algorithm which has feature importance as an attribute or using PCA. Eliminating unimportant features will improve the performance of the models to be tested in the subsequent stage of the analysis. After preprocessing, a set of classification algorithms will be fitted and ranked against the performance of the benchmark and target scores. The top performers will then pass to refinement, here the models will be tuned and scored towards achieving the target score.

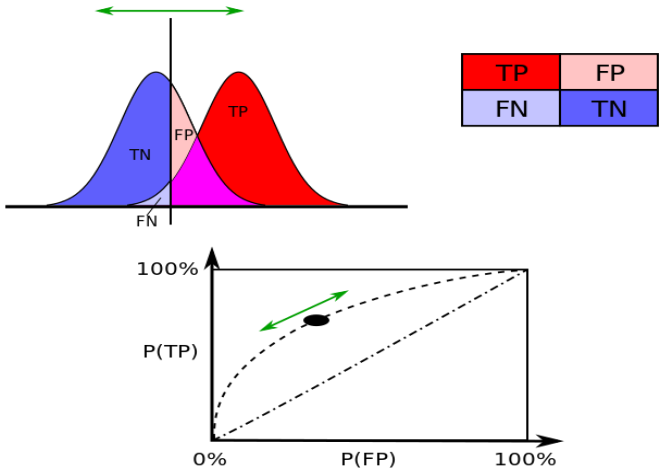
Metrics

The evaluation metric provided for this this competition by Kaggle was the area under the Receiver Operating Characteristic (ROC) Curve. Below is a brief description of how the ROC curve will be applied to measure the data model performance for this problem;

1- The ROC curve is a plot between the True Positive Rate (TPR) and the False Positive Rate (FPR)
2- True Positive means that the model predicted a Happy Customer correctly
3- False Positive means that the model predicted a Happy Customer incorrectly (the customer is actually not happy)
4- TPR is the ratio between correct positive predictions and all positive samples in the test set
5- FPR is the ratio between incorrect positive predictions and all the negative samples in the test set

Now we have defined the axes of the ROC curve as TPR (x-axis) and FPR (y-axis), I will now clarify how the curve is developed. In this project, the model will be predicting class probabilities, i.e. probability of customer being happy or unhappy. So, we will have 2 normal distributions, one for True Positive predictions, and the other for True Negative predictions. As illustrated in the figure below, the intersection between those distributions

and the vertical line (threshold) indicates the False Positive. Thus, the curve represents the trade-off between True Positive (benefit) and False Positive (cost) with a perfect classification on the top left corner of the space, a random classification on the other hand would land on the diagonal dashed line [3].



As the performance of the model increases, the curve will be pulled above the diagonal line increasing the Area Under the Curve (AUC) which reflects the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one [3]; AUC is calculated by using the following equation:

$$A = \int_{\infty}^{-\infty} \mathrm{TPR}(T)\mathrm{FPR}'(T)\,dT = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(T' > T) f_1(T') f_0(T)\,dT'\,dT = P(X_1 > X_0)$$

Where X_1 is the score of the positive instance, X_2 is the score of the negative instance, and f_0 / f_1 are the probability densities described above [3].

As mentioned earlier, the area under the ROC curve was selected as the evaluation metric for this Kaggle competition by Santander, the reasons behind this choice may include;

1- The ROC graphs are independent of class proportion; this is an advantage because as we will see later, the training dataset provided is imbalanced [15, 17]

2- The ROC AUC metric measures a model's ability to distinguish between 2 classes and can also summarize performance in a single number, this makes it a good choice for comparing between classification models [16]

3- The AUC ROC scoring allows model evaluation independent of the threshold. For this reason, this scoring method is popular on Kaggle competitions. The competitors

can focus on developing good classifiers and leave the threshold selection to the organizers of the competition [17].

II. Analysis

Data Exploration

The training dataset provided for this competition includes 76,020 unique ID's each representing a certain customer. Each customer has 370 anonymous features which may relate to the customer's personal information, bank account details, survey results, etc.

```
train.shape
```

```
(76020, 370)
```

Figure 1: original training dataset shape

The target variable is either;

'TARGET' = 1 (Unsatisfied Customer)
'TARGET' = 0 (Satisfied Customer)

All the 370 features are numeric, and the dataset has no NAN values. The dataset is unbalanced with unhappy customers only at 4% of the complete training set.

The testing dataset has 75,818 unique ID's with 369 anonymous features (no target).

```
test.shape
```

```
(75818, 369)
```

Figure 2: original test dataset shape

The dataset set was screened for duplicate and constant features; 34 features were identified with 1 unique element and 29 features were identified as duplicates of other features.

```
The training set has 34 features with 1 unique element, those features will be dropped.
The training set has 29 features with duplicate elements, those features will be dropped.
NAN values in dataset: False.
```

Figure 3: Initial screening for duplicate and constant features

Exploratory Visualization

It will be difficult to illustrate visualizations at this point since the number of features is large. The nature of this problem requires that data pre-preprocessing needs to be performed at an early stage.

For this reason, I will limit data exploration and exploratory visualizations to what is mentioned above and explore further after dimensionality reduction.

Algorithms and Techniques

To solve this classification problem, I need to develop a model which can predict the class probability of each target class and achieve an AUC score which exceeds 0.826874 on Kaggle's private leader board.

I will take the following steps to achieve this objective;

1- **Pre-processing:** to evaluate feature importance and the possibility of reducing the number of features. If successful, the selected feature characteristics will be examined to check if any transformations are required and to test the impact of those transformations on the cross -validation score.

   1.1- **Algorithm:** the algorithm selected is XgboostClassifier since it is highly effective in dealing with large data, it's parametrization is simple, and its prone to over-fitting. Its is also one of the main algorithms I am planning to use for the predictions [8].

   1.2- **Parametrization:** XgboostClassifier will be tuned on the training set. I will start with some initial parameters (to be clarified below) and tune each parameter against the CV score.

   1.3- **Feature Importance:** XgboostClassifier will be used to identify the most important features. The number of features to be selected will be based on the CV score, several trials will be carried out to find the minimum number of features which can be used for fitting.

   1.4- **Data Exploration & Visualizations:** It is expected that the number of features at this stage will be lower allowing for a more detailed feature analysis

2- **Implementation:** based on the features selected in preprocessing, a set of classification algorithms will be fitted on the training to measure their CV scores. Each model will then predict the target on the test set and the result will be submitted on Kaggle (LB score). Comparing AUC CV and LB scores will help detect overfitting.

   2.1- **Algorithms:** the algorithms selected are Quadratic Discriminant Analysis, Random Forest Classifier, Gradient Boosting Classifier, and Xgboost Classifier. Below is a description of the general approach of each of the algorithms selected;

2.1.1- Quadratic Discriminant Analysis: is a classic classifier with a quadratic decision surface. To use this model as a classifier, we need to calculate the class priors P(y = k), the class means (u_k), and the covariance matrix. For QDA, there are on assumptions that the covariance matrix of each of the classes is identical. The likelihood ratio test is then used to predict if a measurement is from a certain class given a threshold t [18,19];

$$\text{Likelihood ratio} = \frac{\sqrt{|2\pi\Sigma_{y=1}|}^{-1} \exp\left(-\frac{1}{2}(x-\mu_{y=1})^T \Sigma_{y=1}^{-1}(x-\mu_{y=1})\right)}{\sqrt{|2\pi\Sigma_{y=0}|}^{-1} \exp\left(-\frac{1}{2}(x-\mu_{y=0})^T \Sigma_{y=0}^{-1}(x-\mu_{y=0})\right)} < t$$

QDA is a simple algorithm to implement and does not require hyperparameter tuning. This makes QDA a choice for this project, especially in the initial stages of the analysis.

2.1.2- *Random Forest Classifier:* is also an easy to use and flexible algorithm which produces good results without hyperparameter tuning. The random forest classifier is built from an ensemble of decision trees (weak learners) which are merged together (strong learner) to get a more accurate and stable results; each tree is trained independently using a random sample of the data. One key advantage of random forests when compared to decision trees is their ability to overcome overfitting because of the large number of trees in the forest; however, this can also make the algorithm slow and ineffective for real-time predictions [20]. For this project, those limitations should not be a problem since a feature importance step will lower the number of features before training, and the dataset is fixed without any updates.

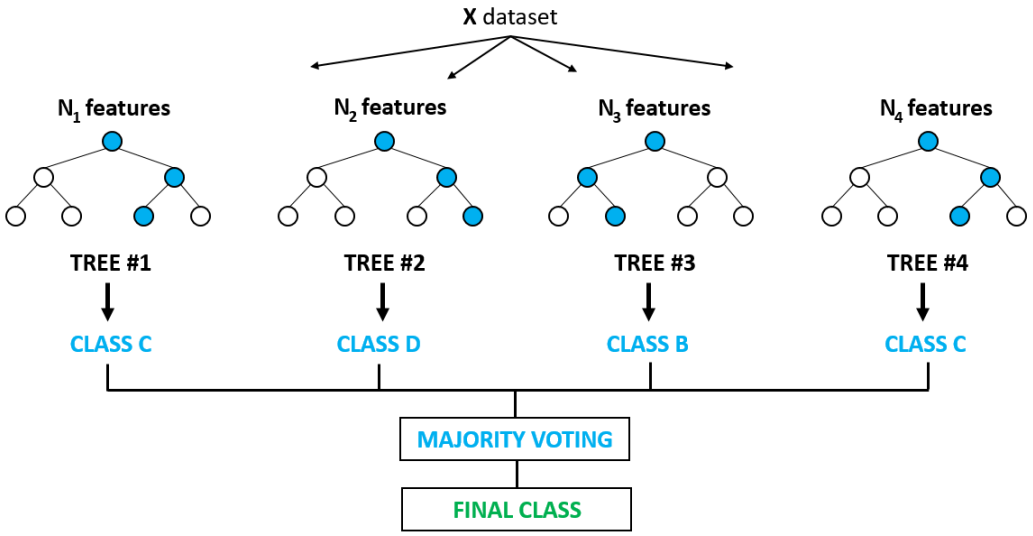


Figure: Random Forest Classifier description

2.1.3- *Gradient Boosting:* in scikit-learn is based on boosting a decision tree function which makes an ensemble learning algorithm which can predict by combing the outputs from individual regression trees. Gradient Boosted Trees differ from Random Forests in the way the trees are built, here the trees are built one at a time where each tree does corrections on the error made by the tree trained before [21]. Since we are dealing with a binary classification problem, that is classifying elements of 2 groups only; the model will be based on a single regression tree which will work towards the optimization of an arbitrary loss function. The tree will be fit on the negative gradient of, in our case, a binomial loss function [22]. Unlike random forests, gradient boosted trees will require more focus towards parameter tuning which in return should bring in higher performance.

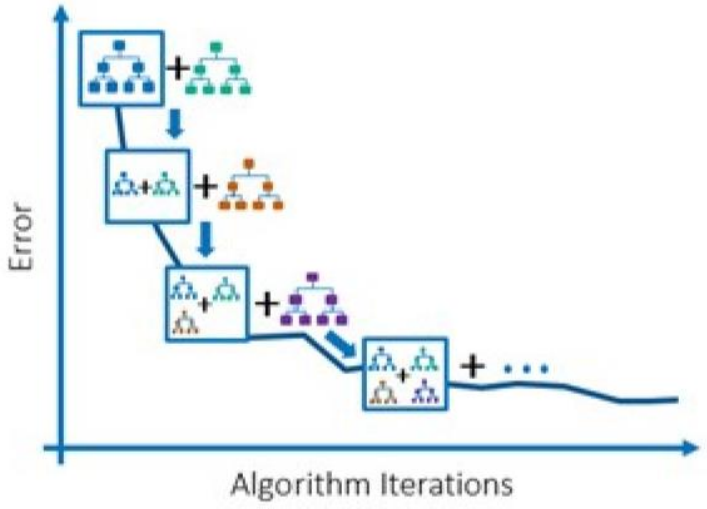


Figure: Gradient Boosting Classifier description

2.1.4- *Xgboost Classifier:* the challenging part of building a decision tree is to decide how to split the current leaf, one way to do it is to take every possible split. This addresses one of the main limitations of gradient boosted trees, and this is what Xgboost can do better by taking the distribution of the features across all data points at a leaf which as a result will reduce the search space of possible feature splits [23]. This is what makes Xgboost a faster algorithm, however, it does have a lot of parameter to tune, mostly to reduce overfitting. In our discussion below, we will make emphasis on which parameters will be selected for tuning.

2.2- **Scoring:** AUC CV and LB scores will be recorded and used to check the performance of each algorithm against the target score.

3- **Refinement:** with the algorithms are fitted and scored, the following steps will take place until the target score is achieved.

   3.1- **Parametrization:** parameter tuning will be applied to the best scoring algorithms (top 2) and the improvements in score will be recorded accordingly

   3.2- **Ensemble:** ensemble models will be developed using the above-mentioned algorithms and the improvements in score will be recorded accordingly

Benchmark

The performance of the data models to be implemented in this project will be measured against the score of a simple linear SGD Classifier, this score gives an indication of the non-linearity of the required solution (low score for SGD). Also, the best model needs to achieve a bronze medal score on Kaggle's Private LB (0.826874).

Benchmark	AUC ROC Score
Linear SGD Classifier score	0.621522
Targeted Kaggle Private LB score	**>** 0.826874

Table 1: benchmark and target scores

III. Methodology

Data Preprocessing

The first step in data preprocessing will be dealing with the results of the initial screening done in data exploration. The duplicate and constant features identified will be dropped for the training set which will result in a new shape of;

```
train.shape
(76020, 307)
```

Figure 4: training set shape after dropping duplicate & constant features

and the same will be applied to the testing set which will now have 306 features;

```
test.shape
(75818, 306)
```

Figure 5: testing set shape after dropping duplicate & constant features

Feature importance will be implemented next for feature selection and dimensionality reduction. An XgboostClassifier will prepared by first fitting the algorithm on the training set using the following initial parameter estimates:

1- learning_rate = 0.01
2- n_estimators = 1100
3- max_depth= 5
4- min_child_weight= 2
5- colsample_bytree= 0.8
6- subsample= 0.8

To ensure good performance, I started with a low learning rate and large number of trees. This combination turned out to be not computationally expensive, so I did not have to lower the number of trees or increase the learning rate. The max_depth can range from 3 to 10, decided to choose something midrange. However, since the dataset is highly imbalanced, a smaller value for min_child_weight was selected. For colsample_bytree and subsample, 0.8 is a common value to start with [9].

The criteria which I will follow to set a threshold for feature selection based on importance will be using the SelectFromModel meta-transformer first to get a general indication of how many features are important. Using the above XgboostClassifier as an estimator for SelectFromModel and then transforming the training set resulted in the following shape;

```
train_SelectFromModel.shape
(76020, 52)
```

Figure 6: SelectFromModel result

This indicates that 52 out of the 306 features explain most of the variability in the data and the rest of the features can be dropped.

The XgboostClassifier was then fitted on the original training set separately to evaluate feature importance, and then the CV score was calculated for the top 50 features, 52 features (as indicated by SelectFromModel), 55 features, and 60 features. This step was done for further verification. The results were as follows;

Number of features selected	CV score
Top 50 features	0.840426
Top 52 features (SelectFromModel)	0.840445
Top 55 features	0.840554
Top 60 features	0.840419

Table 2: Top feature CV scores

The CV score is highest for 55 features, a slight variation from the SelectFromModel result. The training set will be composed of those 55 features and the rest will be dropped.

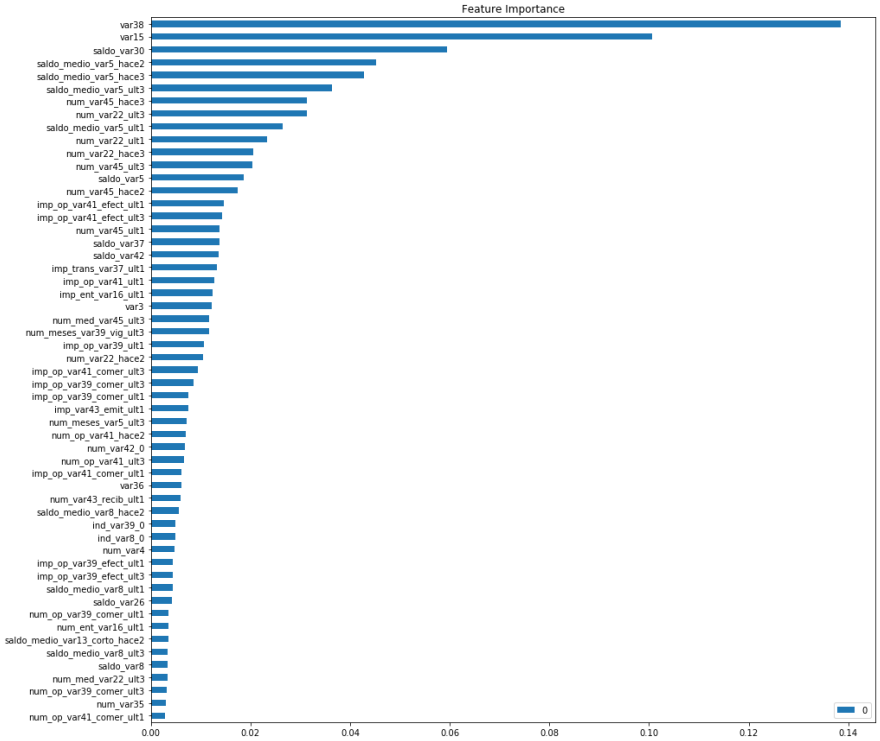


Figure 7: Top 55 features

The above figure illustrates that the first 3 features (var38, var15, saldo_var30) have the highest importance and most probably are the features which will explain most of the variability in the data. For the purposes of this study, I will select the first **6 features** for further preprocessing. The rest of the features will continue to be part of the training set but will only be re-visited for further preprocessing if required (low performance, etc.).

	var38	var15	saldo_var30	saldo_medio_var5_hace2	saldo_medio_var5_hace3	saldo_medio_var5_ult3
count	7.602000e+04	76020.000000	7.602000e+04	76020.000000	7.602000e+04	76020.000000
mean	1.172358e+05	33.212865	1.367967e+04	1579.135311	8.913659e+02	1048.856447
std	1.826646e+05	12.956486	6.301408e+04	12148.452398	9.888597e+03	8189.948852
min	5.163750e+03	5.000000	-4.942260e+03	-128.370000	-8.040000e+00	-476.070000
25%	6.787061e+04	23.000000	0.000000e+00	0.000000	0.000000e+00	0.000000
50%	1.064092e+05	28.000000	3.000000e+00	3.000000	9.900000e-01	2.730000
75%	1.187563e+05	40.000000	2.359950e+02	90.000000	1.221750e+01	83.790000
max	2.203474e+07	105.000000	3.458077e+06	812137.260000	1.542339e+06	544365.570000

Figure 8: Statistical description of the top 6 features

A pair plot visualization of those features will help provide a better picture;

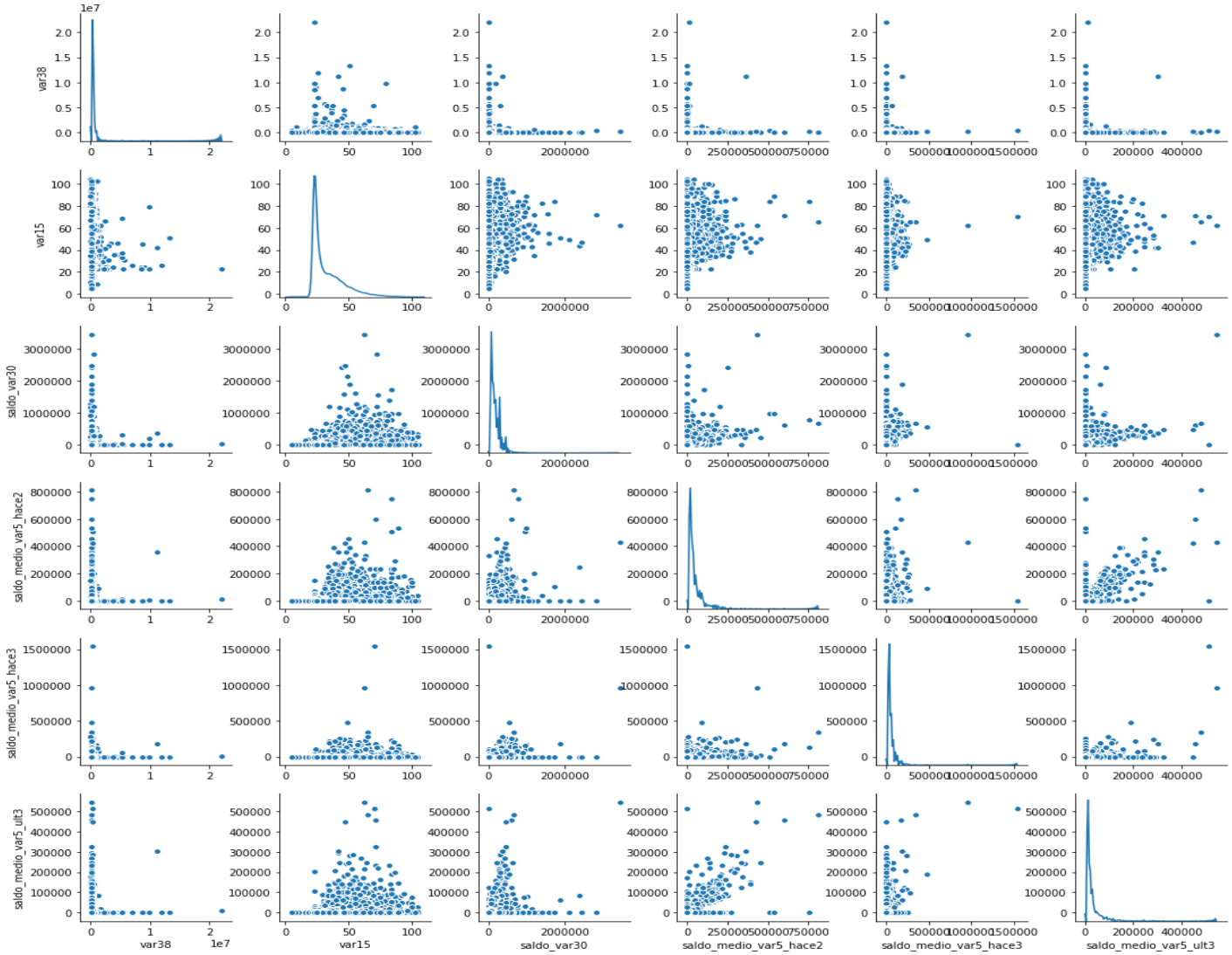


Figure 9: Pair plot presentation of the top 6 features

Starting with the most important feature **var38**, the histograms below of var38 and the logarithm of var38 shows that the distribution is affected by a certain value's high frequency.

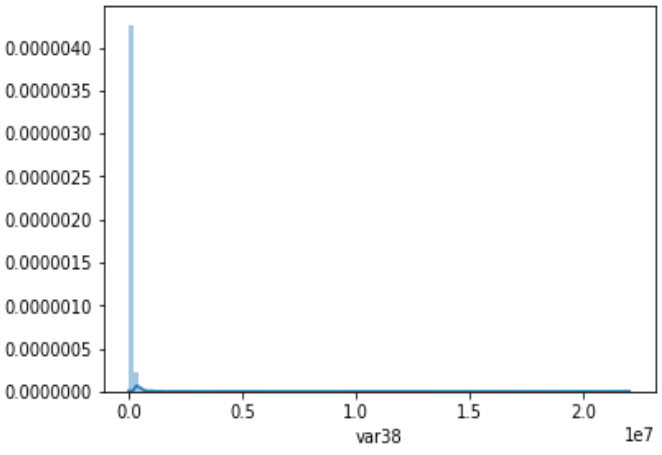


Figure 10: var38 density plot

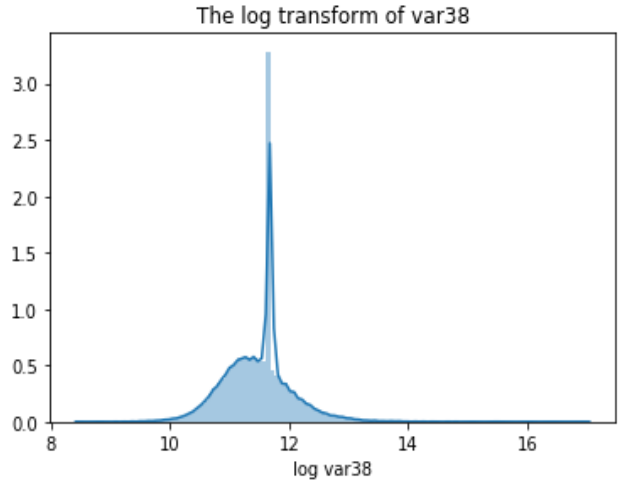


Figure 11: log var38

A value count identifies 117310.97 as the value which is possibly causing this effect, a histogram plot of var38 with this value removed demonstrates why;

117310.979016	14868
451931.220000	16
463625.160000	12
288997.440000	11
104563.800000	11
236690.340000	8
329603.970000	7
104644.410000	7
67088.310000	7
125722.440000	7
128318.520000	7

Figure 12: var38 value counts

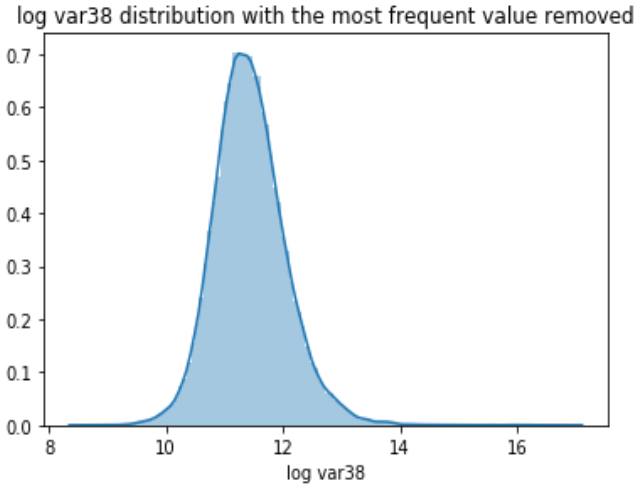


Figure 13: log var38 with 117310.97 removed

A possible way to deal with this spike is by doing the following;

1- creating a feature (**var38_freq_v**) which will be True when var38 has a value of 117310.97~ and False otherwise
2- creating another feature **(var38_log)** which will calculate the log of var38 when var38_freq_v is False and zero otherwise

By splitting var38, the disturbance created by the spike will be smoothed out and information will be preserved (by the Boolean variable).

Now looking at **var15** (fig.14 below), a similar spike exists however to a lower extent when compared to var38. Taking the log transform of var15 was not effective in symmetrizing the distribution or making it more normal, so this variable will be left as-is for further analysis in refinement if required.

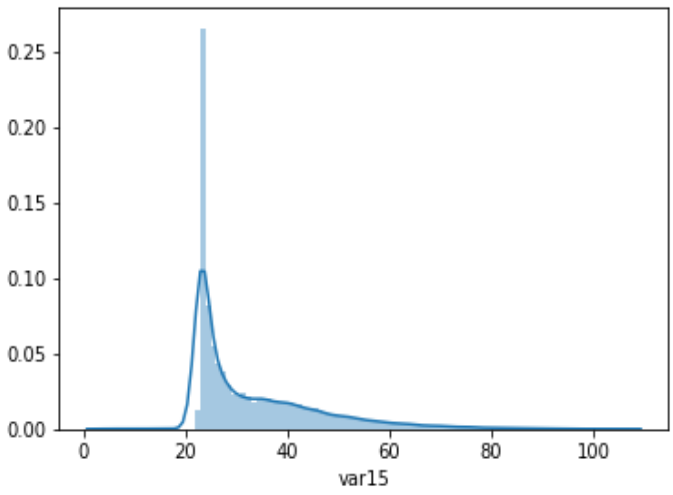


Figure 14: var15 density plot

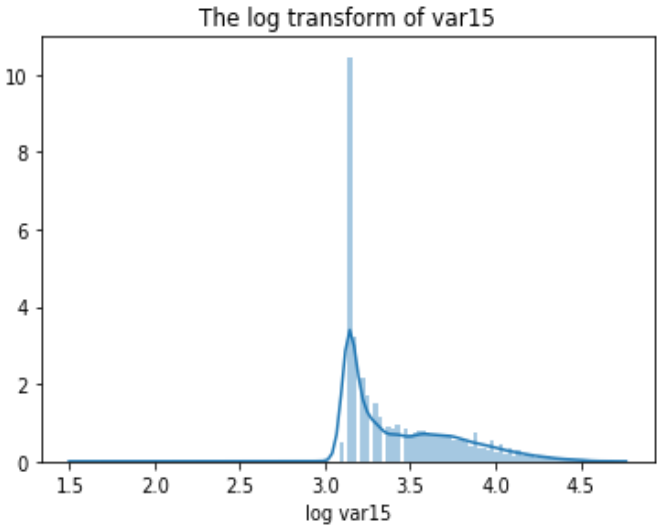


Figure 15: log var15

Referring to figure 9 again (pair plots), it can be observed that features **saldo_var30**, **saldo_medio_var5_hace2**, **saldo_medio_var5_hace3**, and **saldo_medio_var5_ult3** have peaks at zero value. This can be better observed by separate plots and value counts for each of the features;

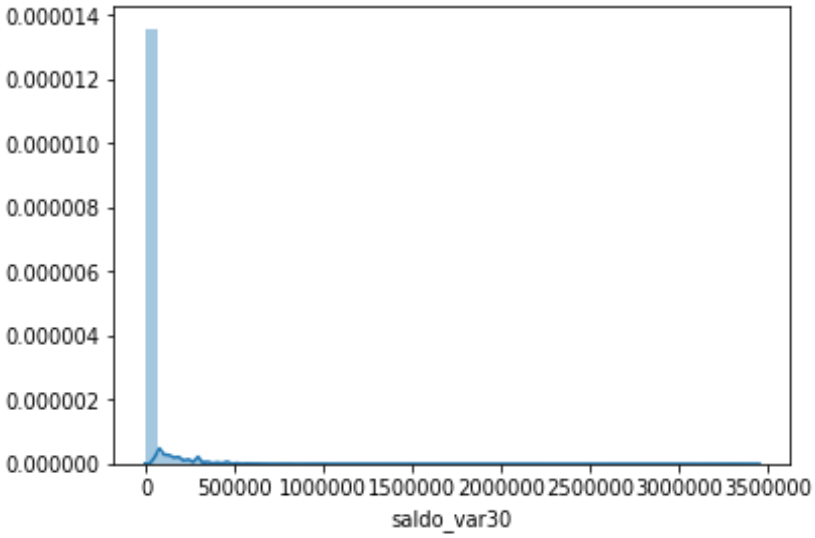


Figure 16: saldo_var30 density plot

0.00	20310
3.00	18290
90.00	5236
30.00	1603
15.00	1477
6.00	926
150.00	823
60.00	686
300.00	400

Figure 17: saldo_var30 value counts

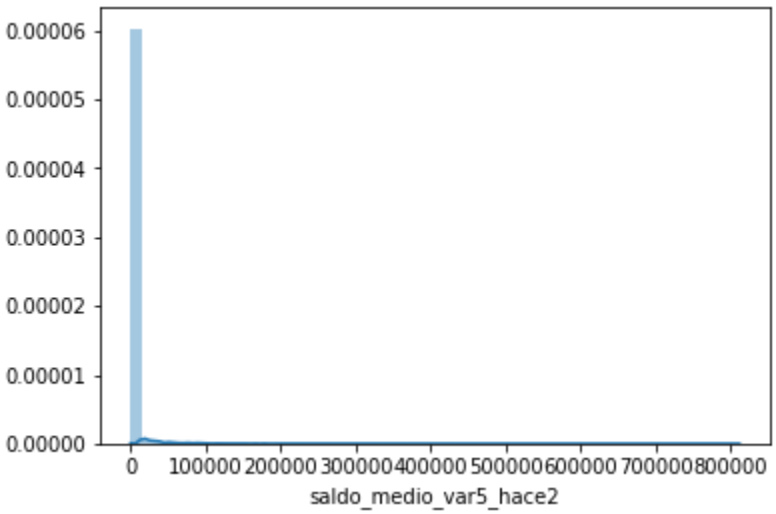


Figure 18: saldo_medio_var5_hace2 density plot

0.00	23241
3.00	18220
90.00	5697
15.00	1513
30.00	1377
6.00	851
150.00	839
60.00	650
300.00	373

Figure 19: saldo_medio_var5_hace2 value counts

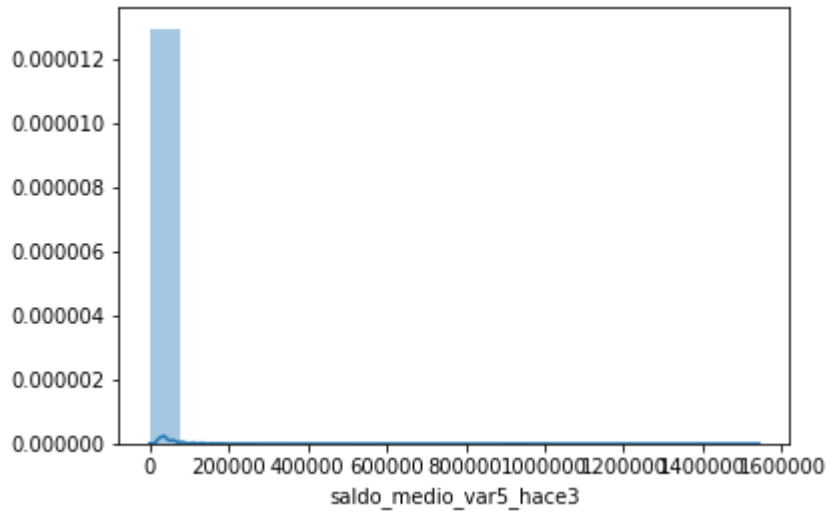


Figure 20: saldo_medio_var5_hace3 density plot

0.00	29637
0.09	992
0.18	875
0.99	741
2.61	703
0.27	629
2.55	600

Figure 21: saldo_medio_var5_hace3 value counts

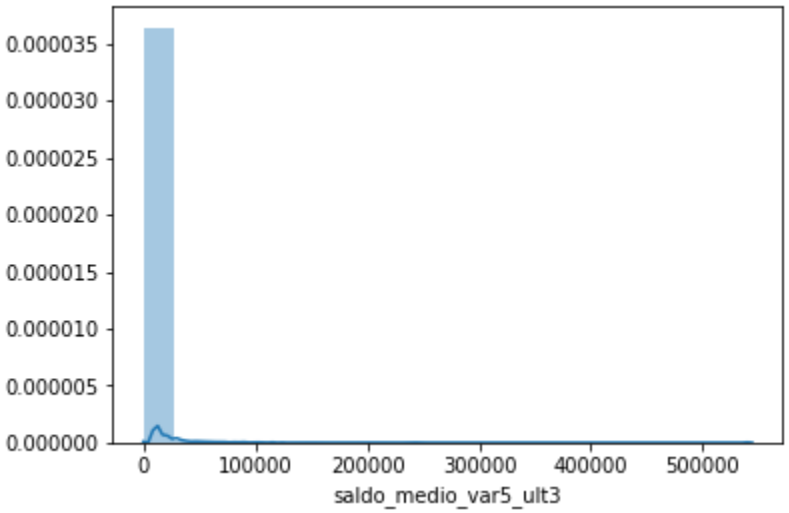


Figure 22: saldo_medio_var5_utl3 density plot

0.00	24664
2.88	1104
2.34	1002
2.85	985
2.07	949
2.79	866

Figure 23: saldo_medio_var5_utl3 value counts

A zero can be indicative of a missing value, or a product code, but it can also be a survey result (satisfaction scale). Since the features are anonymous, we cannot be sure; however, one way to check is to count the number of zeros per row and add this value as a new feature. By doing this, we can see if this feature holds any predictive power within the data model. This step will be done in implementation and the AUC scores with the zero-sum feature and without can then be compared.

Implementation

After preprocessing, the training set is now made of the top 55 features in addition to the var38 feature adjustments (var38_freq_v and var38_log) and the zero-sum feature. Likewise, the test set will be adjusted to include the same additions. Now, the performance of the algorithms selected for this study (Quadratic Discriminant Analysis, Random Forest Classifier, Gradient Boosting Classifier, and Xgboost Classifier) will be measured by

calculating the CV score on the training set and the LB score on Kaggle's test dataset. The LB score is used early on to ensure that overfitting is detected, i.e. an algorithm which scores high on the training set (high CV score) and low on Kaggle's hidden part of the test set (low private LB score) indicates overfitting.

The benchmark linear SGD classifier performed poorly indicating that the problem might be non-linear. Quadratic Discriminant Analysis (QDA) is a nonlinear model which works well when the training set is large, and the number of predictors is smaller than the sample size (rule of thumb $n \geq 5 \times p$) [10]. As illustrated in the table below, the score achieved by QDA (0.723412) is considerably higher than the SGD classifier (0.621522) which demonstrates the non-linearity of the required solution. However, QDA assumes that the predictor variables are drawn from a multivariate Gaussian distribution; and this is not the case as illustrated by several predictor plots in data preprocessing. This surely has a negative effect on the score which should show when using non-linear models that are not based on this assumption.

A random forest classifier does not assume normality and is also non-linear [11], this satisfies the criteria of next model to be tested. The CV and LB scores improved to 0.742084 and 0.735284 respectively, however, still not reaching the target LB score. Another approach for improving the performance of a decision tree is boosting, when gradient boosting was tested on the training set it achieved a significant improvement to reach a score of 0.834210, and an LB score of 0.822190. An important difference between boosting and a random forest is that in boosting, the growth of one tree considers the advancement of other trees and this allows for smaller trees [12]. The parameters set for this gradient boosting run are as follows; min_sample_split will be set to 500 (~0.5% to 1% of the total samples), min_sample_leaf to 50 (starting with a small value to avoid overfitting), max_depth to 8, max_features to square root as a general rule of thumb, and subsample to 0.8 [14].

The last model to be tested is Xgboost which follows the same principle of gradient boosting but with a more efficient approach to how the splits are made at each leaf and more regularization to control over-fitting [13]. Xgboost is a well-known competition winning model and the scores achieved on the training set and Kaggle's test set were the highest compared to the previous models (score table below). The parameters used for Xgboost are fixed to what was explained in feature importance above.

Algorithm	CV Score	LB Score
Quadratic Discriminant Analysis	0.733564	0.723412
Random Forest Classifier	0.742084	0.735284
Gradient Boosting Classifier	**0.837303**	**0.824486**
Xgboost Classifier	**0.840453**	**0.826257**

Figure 23: Algorithm CV & LB scores

* Benchmark 1 (Linear SGD Classifier) LB score: 0.621522
* Target Kaggle LB score: > 0.826874

The highest score achieved (0.826257) by Xgboost is close to the target score of 0.826874 (minimum bronze model score). However, due to the large number of contestants, the score is not enough. In the next section, I will start tuning the best 2 models to push the scores closer to the target. Another reason for tuning the gradient boosting algorithm is to use it again with Xgboost in an ensemble model incase hyperparameter tuning was not sufficient on its own.

Another model I planned to try was support vector machines, however, it was computationally expensive especially when training to predict probabilities. The number of features was reduced to the top 3 most important features, but this also was not sufficient to train the model let alone the time it will take with cross validation. As previously noted, feature selection plays a pivotal role in solving this problem and presents a big part of the analysis; so, some care needs to be taken when following the steps which produced the above-mentioned results (figure 23: Algorithm CV & LB scores). Below are some of the key drivers influencing those results;

A- Feature selection method: several tools could be used to identify the most important features such as *SelectFromModel* from sklearn's feature selection library, *PCA* to reduce the number of dimensions, or using an algorithm which fits the requirement and has a feature importance attribute. In this analysis, an Xgb classifier was used, however, it was observed that different methods could yield different results. When PCA was applied, many dimensions were needed to cover the variability in the data; and for this reason, a more interpretable approach was followed instead. SelectFromModel, gradient boosting, and Xgb have produced similar results, as of the most optimum number of features to keep, with some variations to the ranking of the top features. The effects of such variations will show more in refinement and could be the reason behind changes in the outcome of the analysis.

B- Feature importance algorithm: two were associated with selecting the algorithm responsible for identifying the most important features. First, it is hard to compare apples to apples when it comes to a set of untuned algorithms; some algorithms will produce good results with default parameters, while other algorithms could outperform those results if tuned. The findings of preprocessing in our case are based on the features selected by an Xgb classifier; and the parameters used were clarified earlier in preprocessing. However, Xgb is an algorithm with many parameters mainly to control overfitting, changing the parameters for any reason (e.g. tuning) could also affect the outcome of the analysis.

C- *Visual presentation:* only the top 6 features were plotted, it was difficult to analyze all the top features visually in preprocessing (55 features), and the modifications (var38_freq_v, var38_log, zero's) were the result of the tool

selected. However, if PCA was used as the feature selection method, the analysis could have taken another direction, and the results may also have been different.

Refinement

Parameter tuning for the GradientBoosting Classifier:

The score achieved by **gradient boosting** in implementation [0.834210] was based on default parameters;

loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3,	min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto')

Table 24: Gradient Boosting Classifier default parameters

Out of the complete 18 parameters, the following will be selected for further tuning [14];

- **learning_rate:** will start with a high learning rate to tune the set of parameters below, and then the learning rate will be reduced as a final step. Lower learning rates can be computationally expensive.
- **n_estimators:** the number of sequential trees needs to be tuned because at higher values, gradient boosting can overfit the data.
- **max_depth:** the max depth of a tree also needs to be controlled to avoid overfitting
- **min_samples_split:** this parameter defines the minimum number of samples required in a node to be considered for splitting, this needs to be tuned to avoid the gradient boosting learning characteristics which are too specific (overfitting)
- **min_samples_leaf:** the minimum number of samples required in the terminal node
- **max_features:** the number of features to be considered for a best split
- **subsample:** The fraction of samples to be used for fitting the individual base learner

The function below employs grid-search with cross-validation to tune each of the parameters;

```
score_metric = 'roc_auc'

def model_params(train, test, clf, params):
    grid_obj = GridSearchCV(clf, params, scoring=score_metric, cv=3, n_jobs=-1)
    grid_fit = grid_obj.fit(train, test)
    best_clf = grid_fit.best_estimator_
    best_clf_params = grid_fit.best_params_
    return best_clf.__class__.__name__, grid_fit.grid_scores_, grid_fit.best_params_, grid_fit.best_score_
```

Figure 25: parameter tuning function

To detect overfitting, the CV and LB scores will be recorded and compared for each of the parameter tuning steps. The results for GradientBoosting Classifier tuning are as below;

Parameter	Tuning Range	Result	CV Score	LB Score
n_estimators	40 ~ 120	60	0.838520	0.824930
max_depth	2 ~ 12	9	0.840307	0.825862
min_samples_split	200 ~ 1200	800	0.840307	0.825862
min_samples_leaf	30 ~ 70	50	0.840307	0.825862
max_features	3 ~ 15	7	0.840307	0.825862
subsample	0.6 ~ 0.9	0.8	0.840307	0.825862
learning_rate	0.01	0.01	0.828614	0.814839

Table 26: Gradient Boosting Classifier parameter tuning results

As observed above, two of the parameters were responsible for the improvement in score for GradientBoosting (n_estimators, max_depth). The improvements in score related to the rest of the parameters were too small to be recorded, lowering the learning rate even reduced the score. When comparing with the same model before refinement, we can see that the CV score has improved from 0.837303 to 0.840307 and the LB score has improved from 0.824486 to 0.825862. However, the scores achieved do not meet the required target LB score of 0.826464. Next, the Xgboost Classifier will be tuned to check if the target LB score can be achieved.

Parameter tuning for the Xgboost Classifier:

The default Xgboost parameters will be used here rather than the initial parameters used in implementation. The lower learning rate (0.01) and large number of trees (1100) used in implementation for feature selection turned out to be computationally expensive when used for tuning. For this reason, I will start with the default parameters and then lower the learning rate and increase the number of trees only after tuning the rest of the parameters.

Parameter	Tuning Range	Result	CV Score	LB Score
max_depth	3 ~ 10	5	0.838818	0.822775
min_child_weight	2 ~ 8	4	0.839324	0.823181
gamma	0 ~ 0.4	0	0.839324	0.823181
colsample_bytree	0.4 ~ 0.7	0.6	0.839359	0.825191

subsample	0.5 ~ 0.9	0.7	0.839892	0.825196
n_estimators + learning_rate	1100		0.841007	0.826609
	0.01			

Table 27: Xgboost Classifier parameter tuning results

Tuning the Xgboost Classifier improved the scores further towards the benchmark score of 0.826874; however, neither of the above tuned algorithms achieve the benchmark score. A potential solution to this problem can be to employ an ensemble voting classifier which combines the predictions of the 3 classifiers by voting.

Two Xgboost classifiers will be used together with one GradientBoosting classifier. In the first run, the default parameters will be replaced by the tuned parameters. Next, some of the parameters will be slightly varied to check how this effects the ensemble model performance.

Ensemble model no.	**Algorithm**	**Parameters**	**CV score**	**LB score**
1	GradientBoosting	tuned (table 26)	0.840365	0.825645
	Xgboost	tuned (table 27)		
2	GradientBoosting	tuned (table 26)	0.840844	0.826398
	Xgboost_1	tuned (table 27)		
	Xgboost_2	**Parameters changed:** max_depth = 6, gamma=0.2, colsample_bytree=0.8, subsample=0.8		
3	GradientBoosting	tuned (table 26)	0.841570	0.827330
	Xgboost_1	**Parameters changed:** min_child_weight=2, colsample_bytree=0.8, subsample=0.8		

	Xgboost_2	**Parameters changed:** max_depth = 6, gamma = 0.2, colsample_bytree=0.8, subsample=0.8		

Table 28: Ensemble model results

As observed above, the ensemble model has shown considerable improvement in score when using one gradient boosting classifier with 2 Xgboost classifiers. By slight variations to some parameters, the third ensemble model has successfully exceeded the benchmark score of 0.826874. Increasing max_depth from 4 to 6 gave the model more complexity and since both the CV and LB scores has improved together, overfitting was not an issue in this case. Also lowering the min_child_weight from 4 to 2 in one of the Xgboost classifiers has resulted in more partitioning thus allowing the classifier to capture more variability in the data. Increasing colsample_bytree and subsample to 0.8 also improved the result given the unbalanced nature of the sample. Increasing gamma to 0.2 balanced the larger max_depth and lower min_child_weight parameters in terms of overfitting.

IV. Results

Model Evaluation and Validation

The final model described in refinement consisted of an ensemble of 3 algorithms and has achieved the objective of this project. Since this project is based on a Kaggle competition problem, the required solution is clearly defined through an unlabeled test set which needs to be labeled and submitted to Kaggle for scoring.

In implementation, the performance of each algorithm was measured against both the CV and the LB scores. This was done to ensure the model's robustness in its ability to generalize well to unseen data. Improvements in the CV score alone can be misleading because of the model's tendency to overfit when continuously being tuned and tested on the same dataset (cross-validation). One way to overcome this is to predict and submit Kaggle's unlabeled test after each optimization step and then examine the movements in the private LB score with respect to the CV score. Improvements in both scores together demonstrated the model's ability to predict the labels of a test set which was hidden from the model. The same method was used when tuning the individual algorithms before being combined into an ensemble model, and the improvements in both scores also verified that the parameters were aligned in the right direction.

Justification

The benchmark selected for this project was a score on Kaggle's private leaderboard which needs to rank this ensemble model within medalists. The private leaderboard score is based on the complete test set, this makes the benchmark a good indicator for finding a solution to the problem. As mentioned above, each algorithm was tuned and tested in each step; the CV and private LB scores were recorded to ensure that improvements in score do not relate to overfitting. The successive improvements in score in refinement, as well as the final model exceeding the benchmark score reflects that the model provided a significant solution to the problem.

V. Conclusion

Free-Form Visualization

One of the important features of the customer dataset provided was the large number of features for each customer. It was very important to check how many of the features were duplicated or correlated with the solution. With reference to Figure.7 (top 55 features), the visualization has illustrated that only 3 features are responsible for a great deal of the variability in the data. This indication raised my attention to carryout feature selection which will make training and tuning much less computationally expensive.

Moreover, the histogram visualizations of the most important six features in data preprocessing exposed very important feature characteristics which required processing. For example, the peak at zero in the histogram visualizations was the insight behind adding a zero-sum feature. Also, var38 histogram visualization illustrated the high occurrence of one value, this disturbed the normality of var38's distribution and was resolved by adding a variable with a value conditional on the high frequency value.

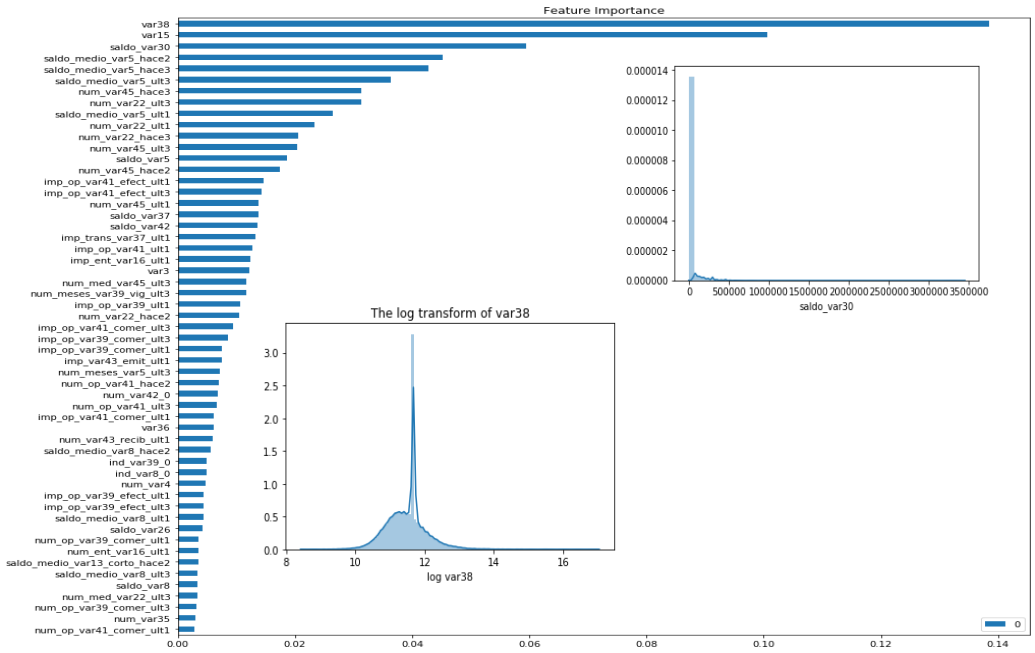


Figure 29: Examples of the important feature visualizations

Reflection

The nature of the problem dictated the process of finding a solution. In the first steps of inspecting the dataset, it was obvious that there were too many customer features. Continuing to the next step of implementation with the same dimensions proved to be computationally expensive, especially in refinement. Even after selecting the top 55 features, carrying out data preprocessing turned out to be very challenging; for example, visualizing 55 features to identify the required transformation was time consuming. So further feature selection was needed to reduce the number of features to only 6 to allow for a deeper examination. Analyzing the top 6 features gave insights to the nature of the complete dataset and resulted in some transformations which were behind part of the final model (zero-sum variable, log var38 transformation). To avoid losing information, the training set used after preprocessing for tuning included the top 55 features but with the preprocessing step changes.

Tuning the GradientBoosting Classifier did not bring significant improvements in each step of GridSearch tuning, the score only improved very slightly for each run. This created the need to explore the performance of Xgboost as an alternative solution. Tuning the Xgboost Classifier presented challenges too. A low learning rate and a large number of estimators resulted in GridSearch CV taking long time durations, for this reason the rest of the parameters were tuned first and the n_estimators/ learning rate parameters were adjusted in the final step.

Although the tuned Xgboost performance was very close to the performance of GradientBoosting, reducing the learning rate and increasing the number of estimators for Xgboost helped achieve the highest score, however, this score did satisfy the objective of this project (exceeding the benchmark). At this point, it was either to re-visit data preprocessing to look for more transformations, or to move forward with an ensemble model using the same tuned algorithms. It was more reasonable to start with the ensemble model first using the 2 already tuned algorithms, the score achieved was close to Xgboost best score. Next, an additional Xgboost classifier was added to the ensemble but with slight changes to the parameters, this improved the score further. The rationale behind the 3rd and final ensemble model parameter changes as explained in the final part of refinement was to overcome overfitting and make the model more generalizable to the unseen part of the test set (hidden by Kaggle). The final model AUC score (0.827330) exceeded the benchmark score (0.826874) and ranked my submission higher than expected. The required benchmark score was to achieve a top 500 rank out of 5,123 participants, however, the score achieved placed the solution at the 126th position.

Improvement

I will now discuss the areas of improvement for each section of the analysis, this will help point out the limitations with enough detail to make them easier to tackle when revisited in the future. First, the selected parameters of the Xgboost classifier used in feature selection can be reviewed, then the tuned Xgboost classifier can be tested and compared to see if

there are any differences in the result. Additionally, only 6 features were selected for the detailed analysis, including more features can provide more insights. The 6 features selected for visualization had skewed distributions and the transformations (box-cox) were not successful to normalize them. Other means of adjustment need to be checked to see if the performance of the algorithms can be improved further, this will also allow for transformations on the complete top 55 features data set used in tuning. Finally, the ensemble model can be developed further by tuning the individual algorithms or expanded to include other classification algorithms to capture more variability in the data.

References

[1] Meinzer, Stefan; Jensen, Ulf; Thamm, Alexander; Hornegger, Joachim; Eskofier, Björn M., Can machine learning techniques predict customer dissatisfaction? A feasibility study for the automotive industry, Artificial Intelligence Research, 2017, Vol. 6 (1), pp. 80-90.

[2] Xia GE, Jin WD. Model of customer churn prediction on support vector machine. Systems Engineering-Theory & Practice. 2008 January; 28(1): 71-7.https://doi.org/10.1016/S1874-8651(09)60003-X

[3] Receiver operating characteristic, Wikipedia at "https://en.wikipedia.org/wiki/Receiver_operating_characteristic".

[4] Reichheld F, Sasser WJ. Zero defections: Quality comes to services. Harvard business review. 1990; 68(5): 105-11. PMid:10107082.

[5] Customers, quality and satisfaction, Sustainable Activity, Santander Corporate, https://www.santander.com/csgs/Satellite?appID=santander.wc.CFWCSancomQP01&c=GSAgrupAsset&canal=CSCORP&cid=1278689260115&empr=CFWCSancomQP01&leng=en_GB&pagename=CFWCSancomQP01%2FGSAgrupAsset%2FCFQP01_GSAgrupAsset Informacion_PT10 .

[6] Bryan Yurcan, July 2017," https://www.americanbanker.com/news/what-santanders-latest-bets-say-about-the-future-of-fintech".

[7] Kaggle Competition, Santander Customer Satisfaction: Which customers are happy customers? https://www.kaggle.com/c/santander-customer-satisfaction#description.

[8] Which machine learning algorithm to choose for my problem? 'https://recast.ai/blog/machine-learning-algorithms/2/'.

[9] Complete Guide to Parameter Tuning in XGBoost (with codes in Python) 'https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/'.

[10] Linear & Quadratic Discriminant Analysis 'http://uc-r.github.io/discriminant_analysis'

[11] Prediction intervals for Random Forests, 'https://blog.datadive.net/prediction-intervals-for-random-forests/'

[12] 'An Introduction to Statistical Learning, with Applications in R', James et. Al, p.323

[13] Quote from the author of Xgboost: GBM vs XGBOOST? Key differences? ‘https://datascience.stackexchange.com/questions/16904/gbm-vs-xgboost-key-differences’

[14] Complete Guide to Parameter Tuning in Gradient Boosting (GBM) in Python, ‘https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/’.

[15] Evaluation metrics for binary classification, Sergül Aydöre, ‘http://www.sergulaydore.com/evaluation-metrics-for-binary-classification/’.

[16] AUC: a Better Measure than Accuracy in Comparing Learning Algorithms, ‘http://www.site.uottawa.ca/~stan/csi7162/presentations/William-presentation.pdf’

[17] Useful properties of ROC curves, AUC scoring, and Gini Coefficients, https://luckytoilet.wordpress.com/2018/04/04/useful-properties-of-roc-curves-auc-scoring-and-gini-coefficients/ ‘.

[18] Quadratic classifier, From Wikipedia, ‘https://en.wikipedia.org/wiki/Quadratic_classifier’

[19] Linear and Quadratic Discriminant Analysis, scikitlearn, ‘ http://scikit-learn.org/stable/modules/lda_qda.html#id4’

[20] The Random Forest Algorithm, Niklas Donges. ‘https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd’

[21] Gradient Boosting vs Random Forest, Abolfazl Ravanshad, ‘https://medium.com/@aravanshad/gradient-boosting-versus-random-forest-cfa3fa8f0d80’

[22] 3.2.4.3.5 sklearn.ensemble.GradientBoostingClassifier, ‘http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html’

[23] Gradient Boosting and XGBoost, Gabriel Tseng, ‘https://medium.com/@gabrieltseng/gradient-boosting-and-xgboost-c306c1bcfaf5’.