Capstone Project - Report

Machine Learning Engineer Nanodegree

1. Definition

1.1 Project Overview

This project is mainly focused on Banking and Financial industry Customer satisfaction. As like other industries, Customer satisfaction is very important for any bank to success with given technology revolutions and change in ways of banking.
Unhappy customers rarely voice their dissatisfaction before leaving. So, it's important to understand the customer satisfaction levels in bank and update the products/services accordingly to maintain good market share.

This project will be mainly focused Kaggle Competition hosted by Santander
Please refer the following link to know more about the bank **Banco Santander - Wikipedia**

Data is sourced from Kaggle competition - **Santander Customer Satisfaction | Kaggle**

- train.csv - the training set including the target
- test.csv - the test set without the target

1.2 Problem Statement

The objective of this project is to identify dissatisfied customers early in their relationship with use of machine learning algorithms. Doing so would allow Santander to take proactive steps to improve a customer's happiness before it's too late. Solutions from this project will be feed into Kaggle Competition hosted by Santander

The problem is to predict the probability of unsatisfactory of a customer. This is a binary classification challenge as our goal is to predict either customer is Satisfied or Unsatisfied.

Target Variable as 0 or 1 –

- 0 for satisfied customers.
- 1 for unsatisfied customers

Please note that Kaggle submission requires the probability of unsatisfied customers in test dataset.

The strategy to solve this problem as follows

- Download Train and Test dataset to Amazon Sagemaker notebook instance and explore and preprocess the data
- Using Feature Selection methods to decrease training dataset dimensionality
- Apply binary Classification algorithms such as XGboost and Linear Learner in Sagemaker as XGboost provides a great efficiency, accuracy in prediction and

linear learner is simple to implement and works best if any linear relationships in the data

- Choose best model from these two which optimizes our objective metric
- Based on finalized model, predict probability of unsatisfactory of customers in test dataset provided by Kaggle and submit the solution for evolution.

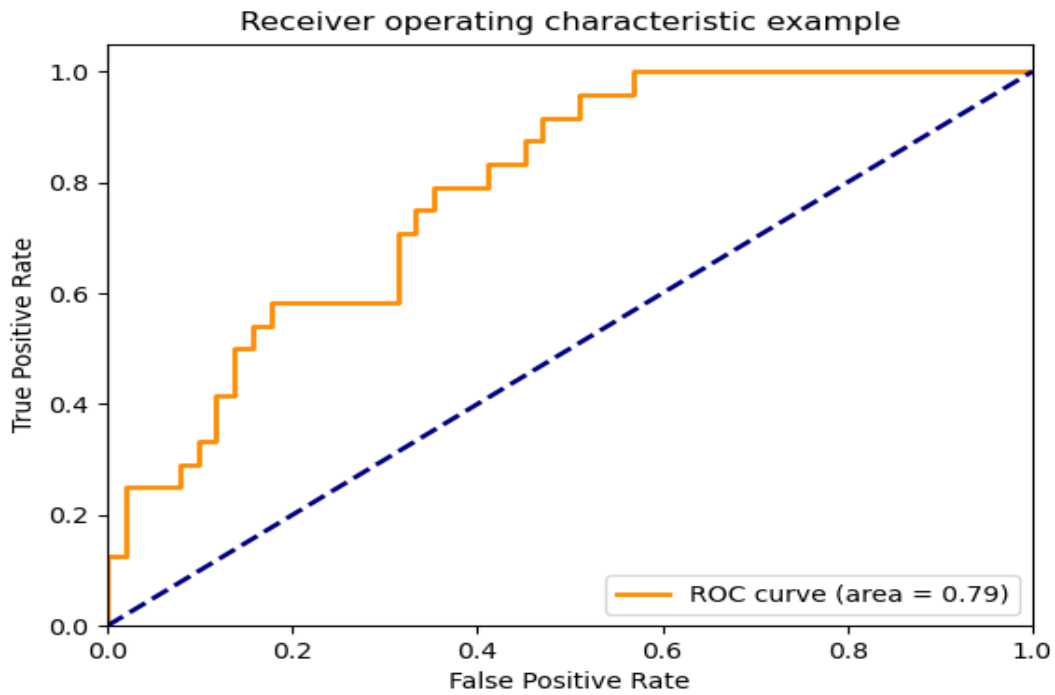
1.3 Evaluation Metrics

Evaluation metrics for the outcome on this project is area under the ROC curve (AUC) as AUC is one of the important evaluation metrics for classification problem as it tells how much our model is capable distinguishing between class. AUC works really well over other classification metrics when data set classes are imbalanced.

Since target class is imbalanced in our dataset, AUC is best suitable metric to evaluate our model performance.

AUC is calculated from a graphical plot curves typically feature true positive rate on the Y axis, and false positive rate on the X axis. This means that the top left corner of the plot is the “ideal” point - a false positive rate of zero, and a true positive rate of one.

Example –



2. Analysis

2.1 Data exploration and visualization

Since this project is based on competition hosted in Kaggle by Santander, we will be using the datasets provided as part of competition which are

i. train.csv - the training set including the target
ii. test.csv - the test set without the target
iii. sample_submission.csv - a sample submission file in the correct format

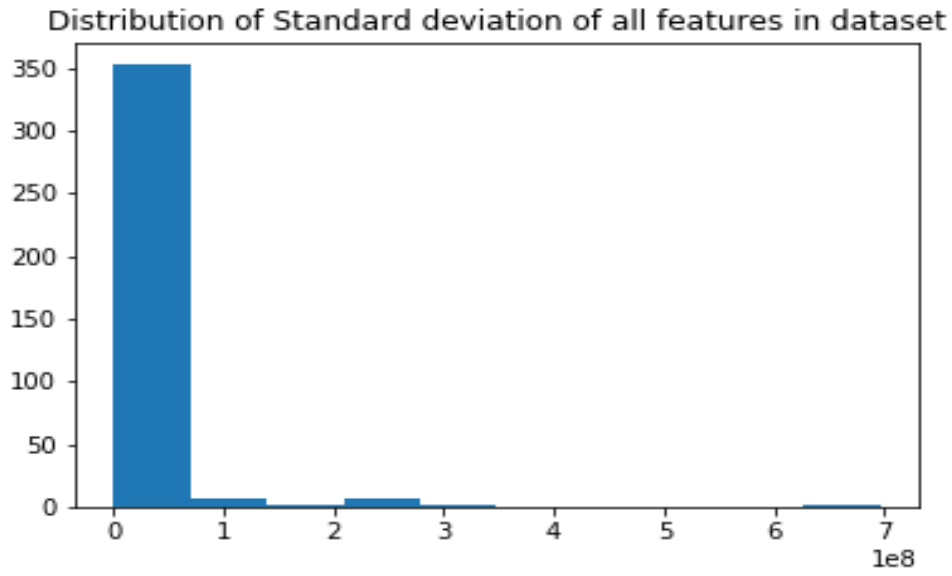
Uploaded this data to notebook instance for the purpose of this project, but datasets can be downloaded from Kaggle URL - **Santander Customer Satisfaction | Kaggle**

There are 370 features in total which are all anonymized (not including the 'TARGET' feature). We will try to understand features based on their distribution

But since there are many features in the dataset, following methods are applied to reduce the dimensions in dataset

2.1.1 - Drop features which are having zero standard deviation

If features have constant values, will drop them from our dataset as these features has zero variance which doesn't add any value to the model predictability power



We can see few features with zero variance from above graph

 # Code snippet to create a list of features with Zero STD to drop
drop_columns = summary_df.iloc[2][summary_df.iloc[2] == 0].index.to_list()

Identified columns to drop – 34 features

['ind_var2_0', 'ind_var2', 'ind_var27_0', 'ind_var28_0', 'ind_var28', 'ind_var27', 'ind_var41', 'ind_var46_0', 'ind_var46', 'num_var27_0', 'num_var28_0', 'num_var28', 'num_var27', 'num_var41', 'num_var46_0', 'num_var46', 'saldo_var28', 'saldo_var27', 'saldo_var41', 'saldo_var46', 'imp_amort_var18_hace3', 'imp_amort_var34_hace3', 'imp_reemb_var13_hace3', 'imp_reemb_var33_hace3', 'imp_trasp_var17_out_hace3', 'imp_trasp_var33_out_hace3', 'num_var2_0_ult1', 'num_var2_ult1', 'num_reemb_var13_hace3', 'num_reemb_var33_hace3', 'num_trasp_var17_out_hace3', 'num_trasp_var33_out_hace3', 'saldo_var2_ult1', 'saldo_medio_var13_medio_hace3']

We have 337 features in our train dataset after dropping the above columns

2.1.2 - Drop duplicate columns

We often find that there would be duplicate columns in datasets provided in Kaggle competitions

Function to find out duplicates in our dataset

```
def dropDuplicates(df):
    '''Function to find duplicate columns in dataframe
       df: dataset'''
    # Declare a list to append the results
    duplicate_columns = []
    # Temp Variable - to store number of columns
    tot_cols = df.shape[1]
    # will reiterate through sequential findings
    for base_col in range(tot_cols):
        col_sel = df.iloc[:,base_col]
        # iterate through all columns
        for compare_with in range(base_col+1, df.shape[1]):
            compare_col = df.iloc[:,compare_with]
            if col_sel.equals(compare_col):
                duplicate_columns.append(df.columns.values[compare_with])
    return duplicate_columns
```

There are 29 duplicate features found in dataset which are

['ind_var29_0', 'ind_var29', 'ind_var13_medio', 'ind_var18', 'ind_var26', 'ind_var25', 'ind_var32', 'ind_var34', 'ind_var37', 'ind_var39', 'num_var29_0', 'num_var29', 'num_var13_medio', 'num_var18', 'num_var26', 'num_var25', 'num_var32', 'num_var34', 'num_var37', 'num_var39', 'saldo_var29', 'saldo_medio_var13_medio_ult1', 'delta_num_reemb_var13_1y3', 'delta_num_reemb_var17_1y3', 'delta_num_reemb_var33_1y3', 'delta_num_trasp_var17_in_1y3', 'delta_num_trasp_var17_out_1y3', 'delta_num_trasp_var33_in_1y3', 'delta_num_trasp_var33_out_1y3']

We have 308 features in our train dataset after dropping duplicate columns

2.1.3 – Check if any features have null values

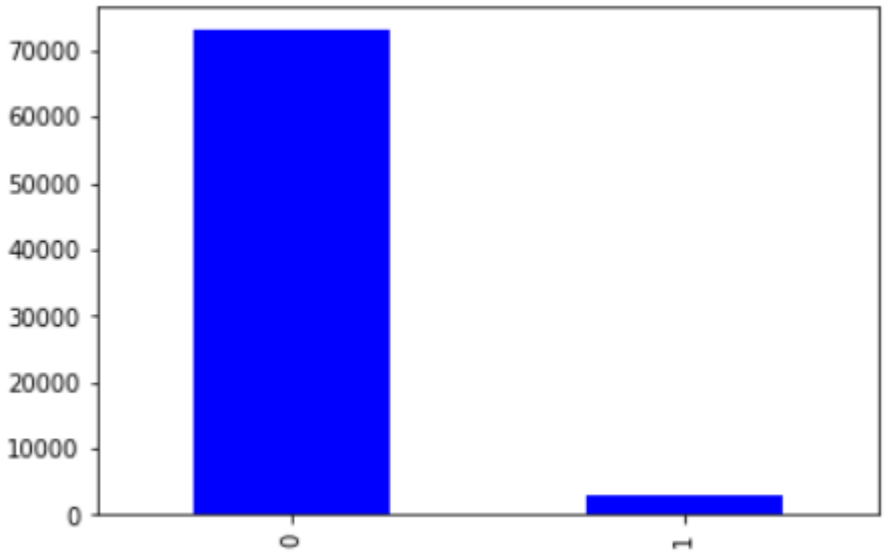
If there are null values in dataset, we either populate the features with reasonable representation such as mean, median..etc or drop features if null value percentage is high

```
# Chekc if any variables has null values
# Store in temporary dataset, as we can't view all columns with Jupyter limitations
Ds = train_df.isnull().sum()/train_df.shape[0]
# Filter only variables with atlest one null value
Ds[Ds > 1]
```

Output is null, so there are no null values for any features in the dataset

2.1.4 – Distribution of target variable

Satisfied Customers percentage – **96.04%**
Unsatisfied Customers percentage – **3.96%**



Our target variable is biased, have only 3.96% of unsatisfied customers in our dataset. Hence, we should factor this in our model development

Since there are c300 features in our training dataset and that too all are anonymized, will explore the variables further after dropping less important variables

2.2 Algorithms and Techniques

We have used Amazon Sagemaker in built algorithms for this project as detailed below

- Linear Learner
- XGBoost

2.2.1 – Linear Learner

Linear models are supervised learning algorithms used for Classification and Regression problems. These models to attempt to model the relationship between target variable and features by fitting a linear equation.

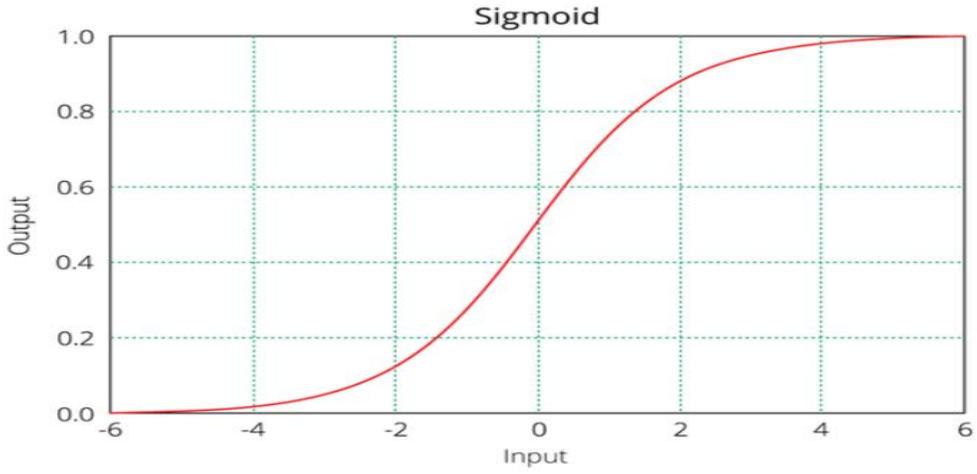
The Amazon SageMaker linear learner algorithm provides solution for regression and classification problems. i.e. For regression problems, Target is a real number.For classification problems, Target is a discrete variable i.e. Spam or not Spam/ Customer is happy or unhappy

How it works for Classification problems:

This algorithm tries to fit our training into a sigmoid function

Sigmoid function - $1/(1+e^{-x})$

This function always yields an output between 0 and 1 and the resultant curve would be like this:



Calculation of Probability once if we fit the logistic regression

$$\ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 x$$

Here
P – Probability of Target occurrence
X – Feature (response variable)

Please find more details of linear learner algorithm and how it works in Sagemaker in the following reference - https://docs.aws.amazon.com/sagemaker/latest/dg/ll_how-it-works.html

Advantages and limitations of Linear Learner:

- Training speed is fast and can customize models for different use cases
- More Explainability power of outcome compare to other models
- Works better if there is good linear relationship in the data.
- If there is no good linear relationship in data, it may not give better results compare to XGBoost

2.2.2 – XGBoost

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost falls under the category of Boosting techniques in Ensemble Learning. Ensemble learning consists of a collection of predictors which are multiple models to provide better prediction accuracy.

In Boosting technique, the errors made by previous models are tried to be corrected by succeeding models by adding some weights to the models.

How boosting works?

In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals.

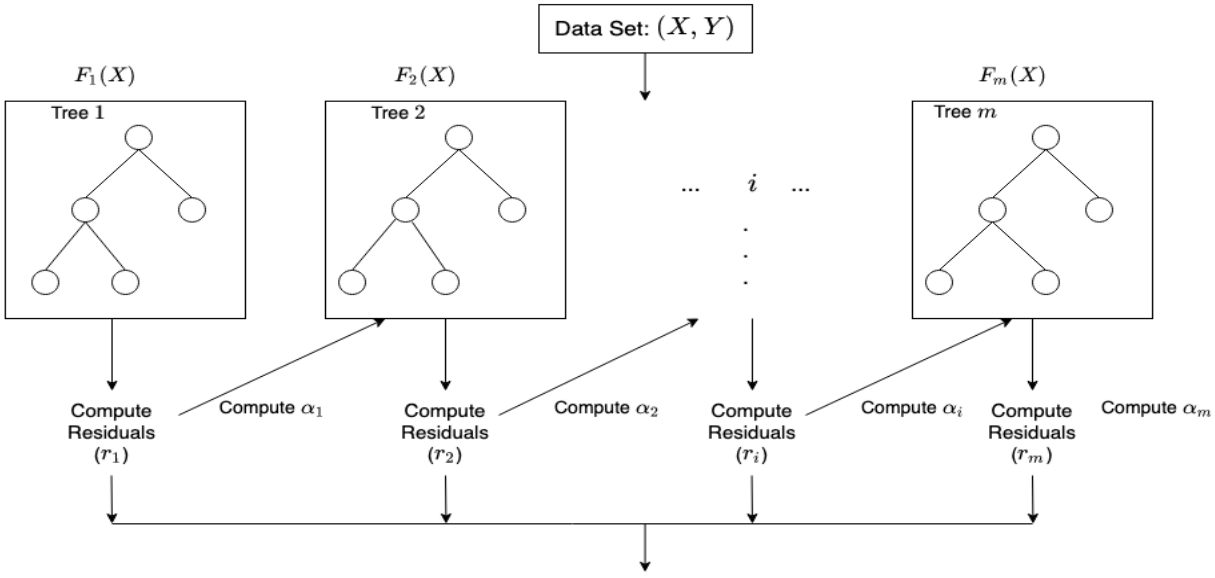
The base learners in boosting are weak learners in which the bias is high, and the predictive power is just a tad better than random guessing. Each of these weak learners contributes some vital information for prediction, enabling the boosting technique to produce a strong learner by effectively combining these weak learners. The final strong learner brings down both the bias and the variance

The boosting ensemble techniques follow Four steps:

- An initial model called F0 is defined to predict the target variable y. This model will be associated with a residual (y – F0) i.e. the deviance from actual
- A new model h1 is fit to the residuals from the previous step
- Now, F0 and h1 are combined to give F1, the boosted version of F0. The residual error from F1 will be lower than that from F0
- Again, new model will be built based on learnings from F1. These iterations will continue until residuals have been minimized as much as possible

Below is a brief illustration on how gradient tree bosting works.

(Source – Amazon Sagemaker)



$$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

Please refer the following link to know more details on XGBoost - Reference

Advantages of XGBoost:

1. Regularization - XGBoost has in-built L1 (Lasso Regression) and L2 (Ridge Regression) regularization which prevents the model from overfitting
2. Handling Missing Values: XGBoost has an in-built capability to handle missing values
3. Effective Tree Pruning: A GBM would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a greedy algorithm.

Weakness of XgBoost:

1. Training time is pretty high for larger dataset
2. Explainability is lower compare to linear learner

2.3 Benchmark

We are using a benchmark of at least 50% AUC as Machine learning models in our project are built with aim of good separation power on target class compare to basic random model

3. Methodology

3.1 Data processing - Dimensionality reduction

As there c300 features in our training data, Dimensionality reduction methods play an important role here as training models with c300 features would be associated with lot of cost.

There are many dimensionality reduction methods such as PCA, SelectKbest, Chi..etc which help us to capture the variance in our training data with less features compare to the original set of features.

We would be using Extra Trees Classifier in our project to find out top important features in our dataset as this algorithm is highly efficient in classification problems and associates with less cost

3.1.1 Overview of ExtraTreesClassifier

Extremely Randomized Trees Classifier (Extra Trees Classifier) is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a “forest” to output its classification result. In concept, it is very similar to a Random Forest Classifier and only differs from it in the manner of construction of the decision trees in the forest. (source - geeksforgeeks.org)

3.1.2 application of ExtraTreesClassifier

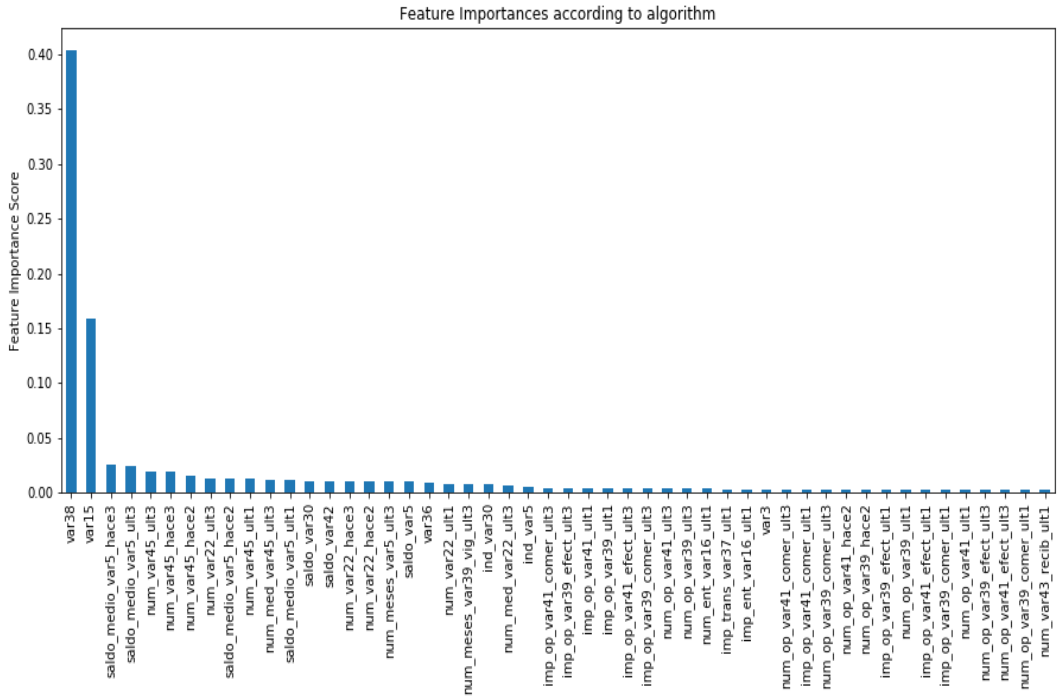
We have referred Sckit-learn package to use this method in our project, more detail at Scikit-learn - Extra trees classifier

Code Snippet

```
## # Feature selection with ExtraTreesClassifier
clf = ExtraTreesClassifier(random_state = 69)
selector = clf.fit(train_df.drop(['TARGET'],axis=1),train_df['TARGET'])

# plot most important features
feat_imp = pd.Series(clf.feature_importances_, index = train_df.drop(['TARGET'],axis=1).columns.values).sort_values(ascending = False)
# Visualise Top 50 features
feat_imp[:50].plot(kind = 'bar', title = 'Feature Importances according to algorithm',
                   figsize = (15, 8))
plt.ylabel('Feature Importance Score')
plt.subplots_adjust(bottom = 0.3)
plt.show()
```

Top 50 features based on importance score returned by Extra Trees Classifier



Let's explore few top important variables to build an understanding on data

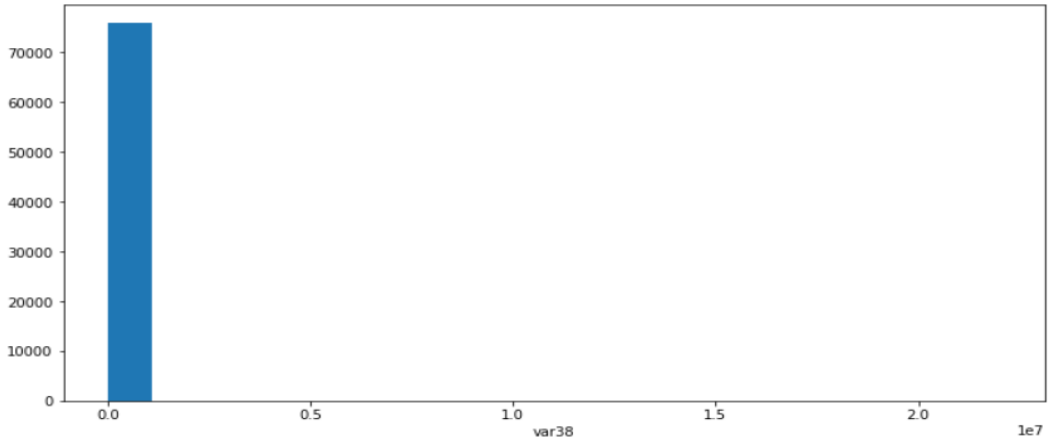
3.1.3 Exploration of top 2 variables based on feature importance score

3.1.3.1 Var38

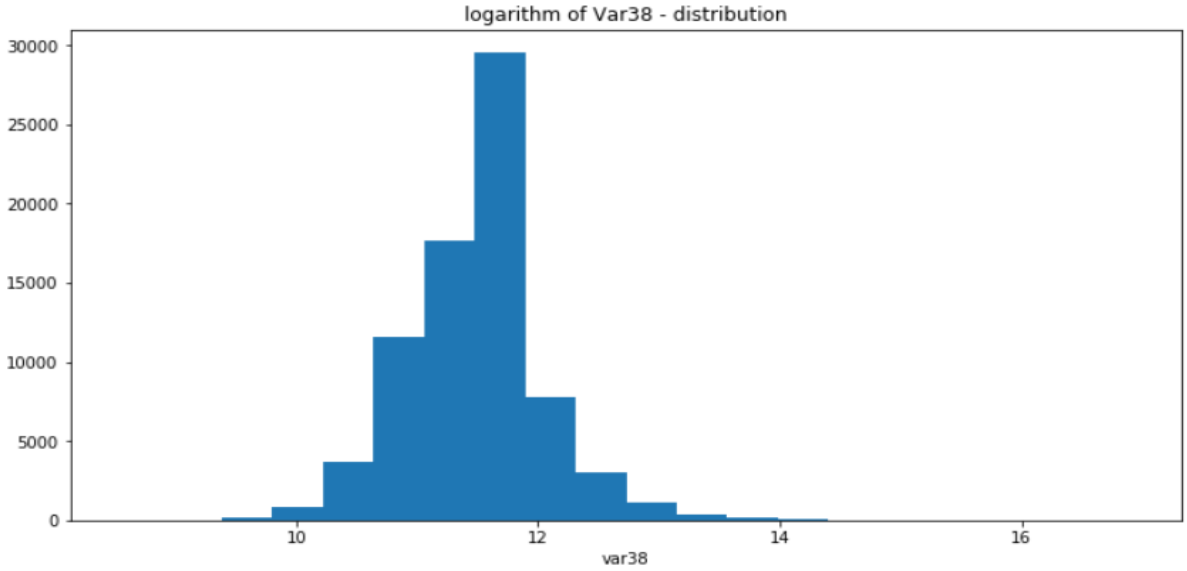
This feature has Standard deviation 182,665 which represents the data is highly distributed.

Here is the distribution of Var38

Minimum of Var38: 5,163.75
Maximum of Var38: 22,034,738.76



Since there are few big values for Var38, we can't view the distribution properly in histogram.
Let's apply logarithm on this feature for better visualization of distribution

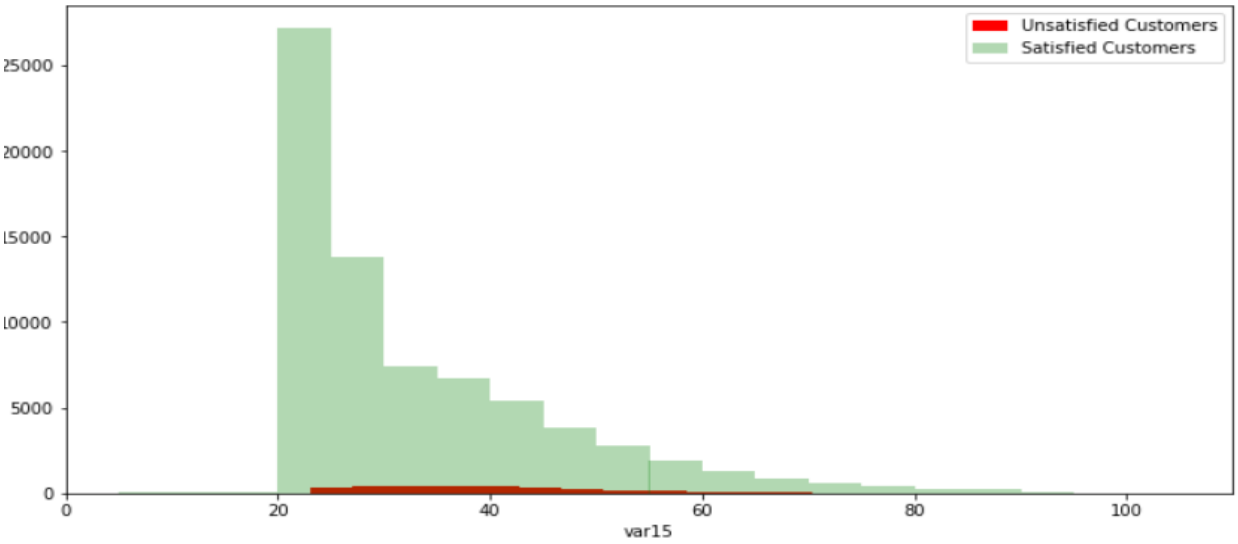


From this distribution and by applying business knowledge, we can understand this variable represents the balance of any banking product such as savings or mortgage...etc

3.1.3.2 Var15

Distribution metrics of this variable

Minimum of var15: 5
Maximum of Var38: 105
Standard deviation of var15: 12.96



Based on distribution, this variable might be representing the age of customer.
With that assumption, we can see less un-satisfaction in age old customers compare to middle age

3.2 Model dataset preparation

Based on feature importance score, we would be using top 40 features for model building as we see the importance score is very less/ going down after that.
The top 40 number is decided based on many experimentations results I.e., AUC

```
# Select top 40 features for model development , as these features collectively explain the decent variance in target variable;
Top_Features = feat_imp[:40].index.to_list()
```

Features finalized for model building

['var38', 'var15', 'saldo_medio_var5_hace3', 'saldo_medio_var5_ult3', 'num_var45_ult3', 'num_var45_hace3', 'num_var45_hace2', 'num_var22_ult3', 'saldo_medio_var5_hace2', 'num_var45_ult1', 'num_med_var45_ult3', 'saldo_medio_var5_ult1', 'saldo_var30', 'saldo_var42', 'num_var22_hace3', 'num_var22_hace2', 'num_meses_var5_ult3', 'saldo_var5', 'var36', 'num_var22_ult1', 'num_meses_var39_vig_ult3', 'ind_var30', 'num_med_var22_ult3', 'ind_var5', 'imp_op_var41_comer_ult3', 'imp_op_var39_efect_ult3', 'imp_op_var41_ult1', 'imp_op_var39_ult1', 'imp_op_var41_efect_ult3', 'imp_op_var39_comer_ult3', 'num_op_var41_ult3', 'num_op_var39_ult3', 'num_ent_var16_ult1', 'imp_trans_var37_ult1', 'imp_ent_var16_ult1', 'var3', 'num_op_var41_comer_ult3', 'imp_op_var41_comer_ult1', 'num_op_var39_comer_ult3', 'num_op_var41_hace2']

4. Model Development and Results

4.1 Splitting data for Validation and Testing

Since our data is biased, we have used stratified split of our train dataset into Validation and test dataset

We have used the following composition of data in our model

- Training – 60% of data
- Validation – 20 % data
- Testing – 20 % data

4.2 XGBoost

We have used two different modelling techniques to predict the probability of unsatisfied customer, details as follows

- Linear learner (Binary- Logistic)
- XGboost

XGboost modelling results as follows

4.2.1 Model training

XGBoost model instance used in our project

```
# Create XGboost model
xgb = Estimator(container,
                role,
                instance_count=1,
                instance_type='ml.m4.xlarge',
                output_path='s3://{}/{}/output'.format(bucket_name, prefix))

xgb.set_hyperparameters( max_depth=5,
                         min_child_weight=1,
                         gamma= 0,
                         subsample= 0.8,
                         colsample_bytree= 0.8,
                         objective= 'binary:logistic',
                         nthread=4,
                         eval_metric ='auc',
                         scale_pos_weight=0.396,
                         seed=4242,
                         num_round=30)
# scale_pos_weight - 0.396 represents % of unsatisfied customers in our train dataset
# after few experimentations - decided with this hyperparameters set
```

```
# Fit XGboost model
xgb.fit({'train': s3_input_train, 'validation': s3_input_validation})
```

Hyperparameters are decided based on experimentation

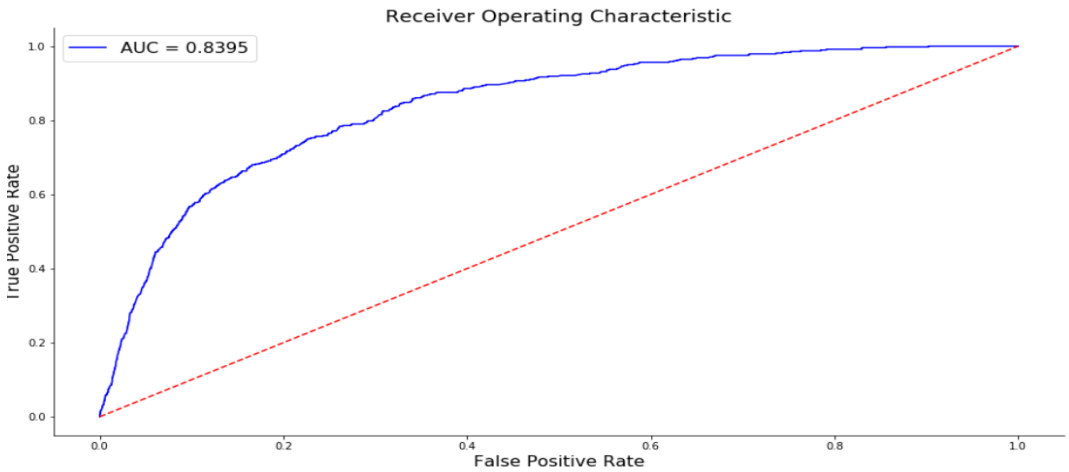
4.2.2 Model deployment and results

Once the model is trained, we have deployed the model to create an active endpoint.

```
# Deploy XGboost model
xgb_predictor = xgb.deploy(initial_instance_count=1,
                           instance_type='ml.m4.xlarge')
```

We have supplied our test dataset to endpoint in batches of 500 records to predict the probability of unsatisfied customer

Results – 83.95% AUC



4.3 Linear learner

4.3.1 Model training

Linear learner model instance used in our project

```
linear = LinearLearner(role=role,
                                train_instance_count=1,
                                train_instance_type='ml.c4.xlarge',
                                predictor_type='binary_classifier',
                                output_path=output_path,
                                sagemaker_session=session,
                                epochs=15,
                                binary_classifier_model_selection_criteria='f_beta',
                                positive_example_weight_mult='auto')
```

We have used 70% of training and 30% test data composition for this model

```
# Convert data into numpy;
train_x_np = X_train_ls.values.astype('float32')
train_y_np = y_train_ls.values.astype('float32')
# create RecordSet
formatted_train_data = linear.record_set(train_x_np, labels=train_y_np)
```

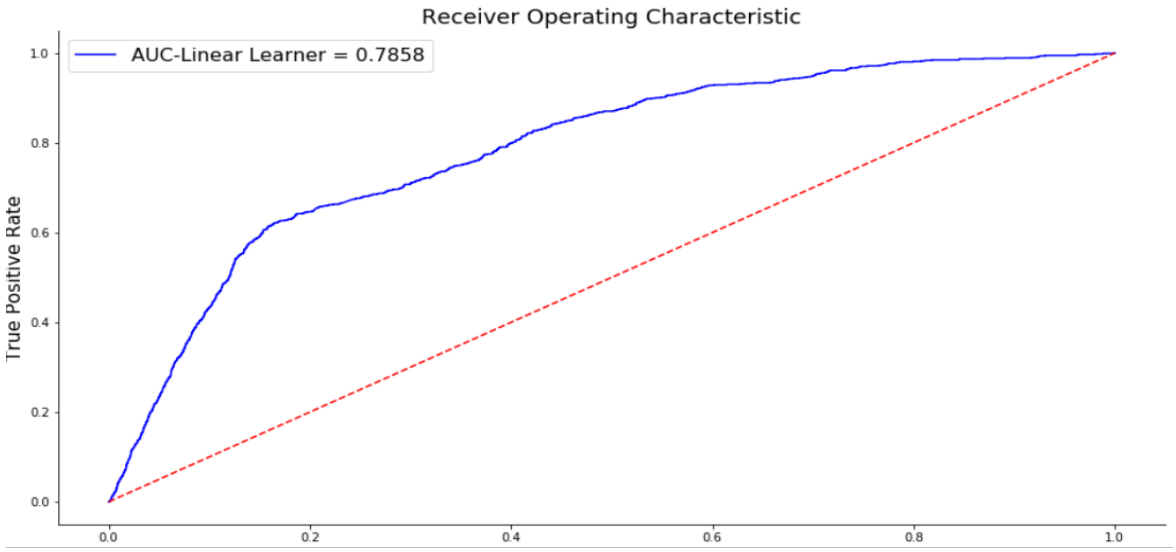
4.3.2 Model deployment and results

Once the model is trained, we have deployed the model to create an active endpoint.

```
%%time
# train the estimator on formatted training data
linear.fit(formatted_train_data)
```

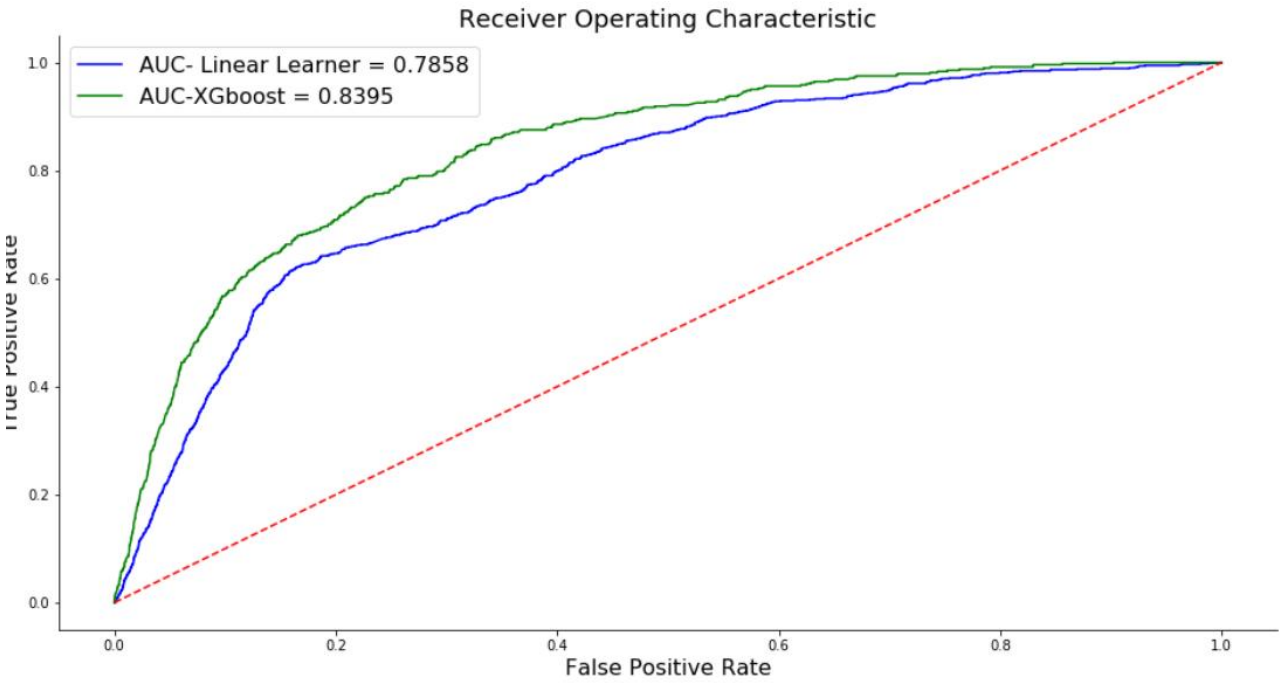
We have supplied our test dataset to endpoint in batches of 100 records to predict the probability of unsatisfied customer

Results – 83.95% AUC



4.4 Justification

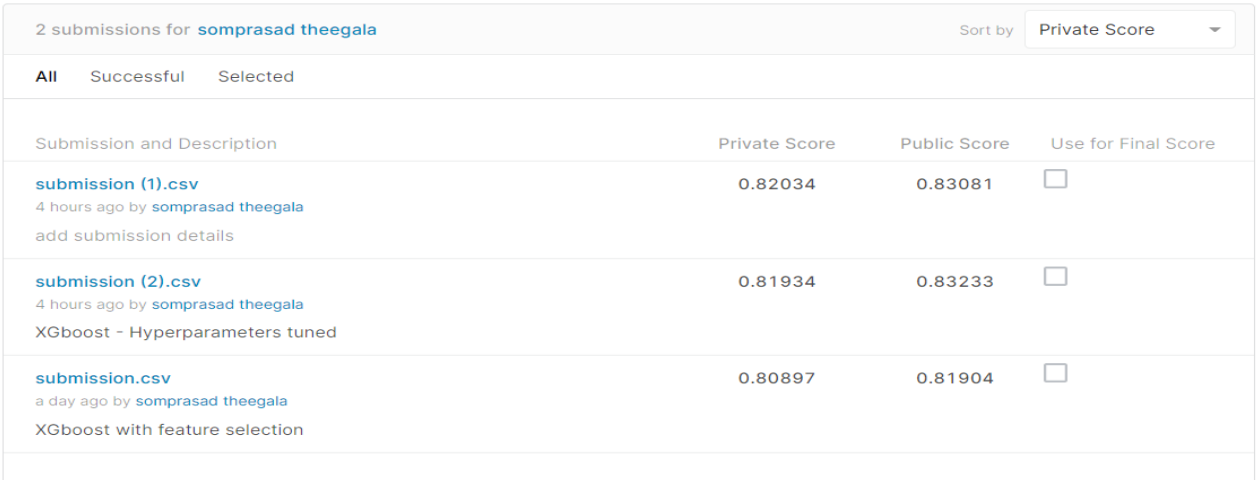
Here is the comparison of both models in terms of our objective metric (i.e. AUC)



XGBoost is delivering the AUC of 83.95% where as linear learner mode is able to produce the AUC of 78.58%. Since XGBoost algorithm is giving us better prediction power over linear learn, we have selected XGBoost as final solution model for our project.

4.5 Submission to Kaggle

Submitted the three solutions to Kaggle with variations on XGboost based on test dataset provided by Kaggle



4.6 Improvements and Next steps

We may attain better accuracy with more powerful hyperparameters tuning on XGboost or application of ensemble techniques on this data. I would be using following the article to tune the parameters further **https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/**

If this would have been real data and project, we would create proper API to return the probability scores from our applications

References

1. Sagemaker Examples
   amazon-sagemaker-examples/introduction_to_applying_machine_learning at master · aws/amazon-sagemaker-examples (github.com)

2. Extra trees Classifier
   sklearn.ensemble.ExtraTreesClassifier — scikit-learn 0.24.1 documentation (scikit-learn.org)