Santander Customer Transaction Prediction

Houssain Alali
501070265
CIND 820 – Big Data Analytics Project Report

Ryerson University

Table of Contents

1. Introduction

The problem presented on the Kaggle competition: Santander's mission is to help people and businesses succeed by providing them with financial products and services to help them achieve their goals. They use data science and machine learning techniques to better understand their customers and identify new solutions to common problems like determining customer satisfaction, predicting product purchases, and assessing loan repayment abilities. They actively collaborate with the global data science community to improve their algorithms in detecting binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan? (Kaggle, 2019)

1.1 Objective

Identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

2. Data Description

The data provided is unlabeled, and contains a numeric target variable (1 or 0), an “id_code” column to identify the sample and two hundred columns (features) named “var_0” to “var _199”. This data comes in the form of a CSV file.

The following table shows a quick view of the data.

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...

Figure 1 - First Look at the Data

Each sample contains:

- 200 numerical features (floating point) labelled “var_0”, “var_1”, … “var_199”
- 1 target class: “target”
- 1 unique sample id code: “ID_code”

3. Data Inspection and Analyses

To gain a better understanding of the features, the initial step is to analyze basic data statistics, considering the absence of information about each feature's representation. It is also important to verify if any data is missing.

This illustrates no missing data in the data frame.

```
ID_code    0
target     0
var_0      0
var_1      0
var_2      0
           ..
var_195    0
var_196    0
var_197    0
var_198    0
var_199    0
Length: 202, dtype: int64
No missing data in dataset
```

Figure 2 – Missing Data Results

The following table shows some of the statistics obtained from the data:

	target	var_0	var_1	var_2	var_3	var_4
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
mean	0.100490	10.679914	-1.627622	10.715192	6.796529	11.078333
std	0.300653	3.040051	4.050044	2.640894	2.043319	1.623150
min	0.000000	0.408400	-15.043400	2.117100	-0.040200	5.074800
25%	0.000000	8.453850	-4.740025	8.722475	5.254075	9.883175
50%	0.000000	10.524750	-1.608050	10.580000	6.825000	11.108250
75%	0.000000	12.758200	1.358625	12.516700	8.324100	12.261125
max	1.000000	20.315000	10.376800	19.353000	13.188300	16.671400

8 rows × 201 columns

Figure 3 – Basic Statistics Extraction Sample

Observation:
The data ranges across the features are not comparable, data pre-processing will be applied to the features using the MinMaxScaler (scikit-learn) in order to bring all values to the [0, 1] range. Also no missing data was found.

Further preliminary analysis is conducted in order to see the ‘target’ distribution.

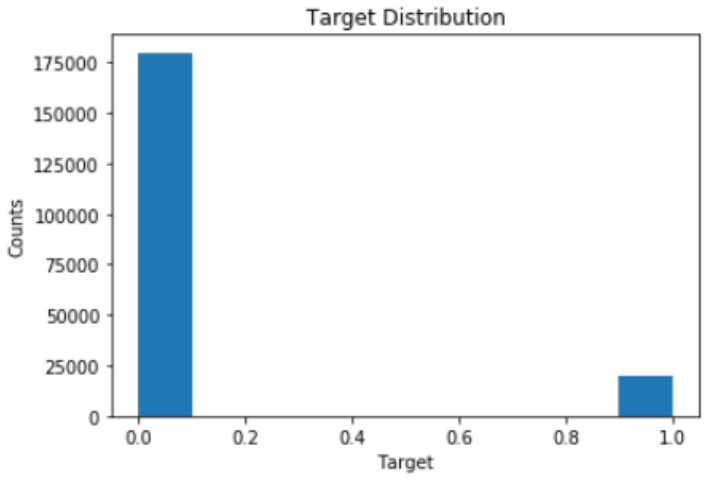

Figure 4- Target Class Distribution

Observation:
Given the data class distribution is not balanced (90% of class 0 and 10% of class 1), stratified sampling is used in order to split a test set from the original training data.

Split of the original data set (200K samples) is done in the following form:

- 80% Training Set
- 20% Test Set

Another implication of this finding is that using accuracy scoring will not accepted due to the skew of classes; therefore the area under the curve (AUC) score is selected for all model training.

Also as part of the preliminary data analysis, plotting of the feature histogram distributions (50 bins) is conducted. The following table shows only a few features.

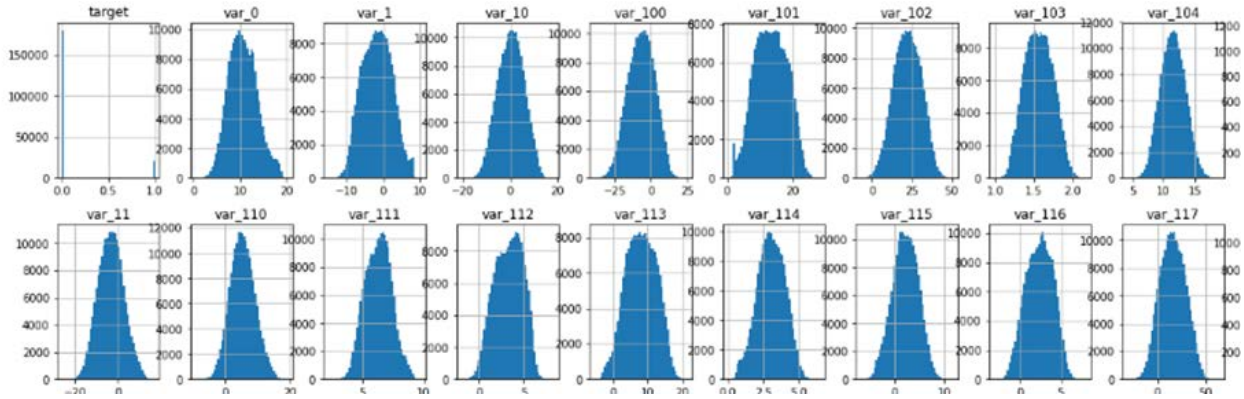

Figure 5 - Data Histograms Sample

Observation:
Upon further analysis, it has been determined that all the distributions follow an approximate "Normal distribution." Therefore, no additional information can be gathered from the distribution analysis results.

After examining the basic data statistics and verifying for missing data, the correlation matrix was extracted and can be referred to as Figure 6 - Data Correlation Matrix. However, the analysis indicates that the features do not exhibit a linear correlation. Therefore, no additional information can be gathered from the results of the correlation matrix.

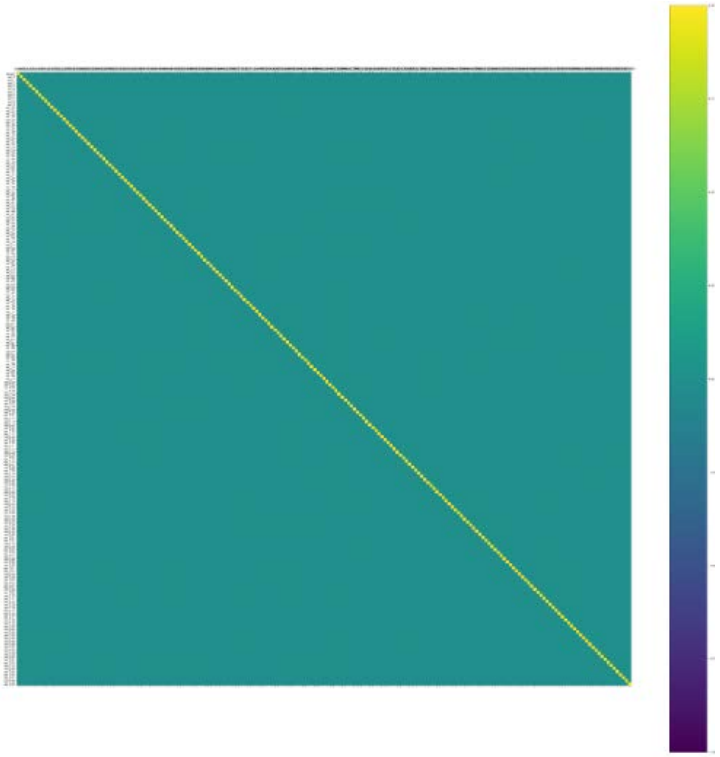

Figure 6 - Data Correlation Matrix

As a final step of the data analysis, Principal Component Analysis (PCA) is conducted to explore the distribution of information among the features. PCA is a statistical technique that can identify underlying patterns in the data by transforming the original features into a set of new, uncorrelated variables called principal components. By doing so, it can help to identify which features contribute the most to the variation in the data and can be useful for data visualization and dimensionality reduction.

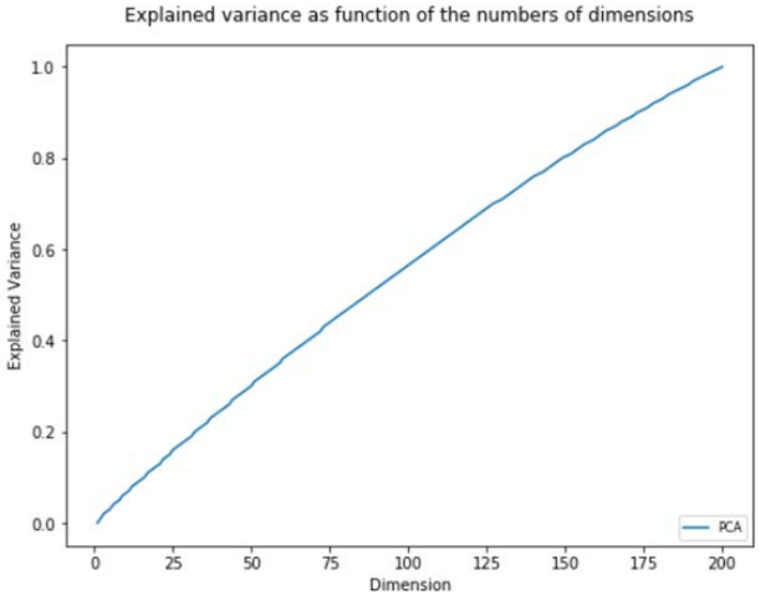

Figure 7 - Explained Variance as function of features

Observation:
After conducting Principal Component Analysis, the graph shows a "linear" distribution of information among all the features. This suggests that there is no significant benefit to performing linear dimensionality reduction techniques, such as Principal Component Regression or Linear Discriminant Analysis. However, it is still possible that non-linear dimensionality reduction techniques may be useful in reducing the number of features while preserving important information, such as Manifold Learning techniques like t-SNE.

4. Dimensionality Reduction

In order to gain more knowledge about the class distributions PCA, LLE and tSNE 2D visualization was attempted to see if there was any visible separation between the classes.

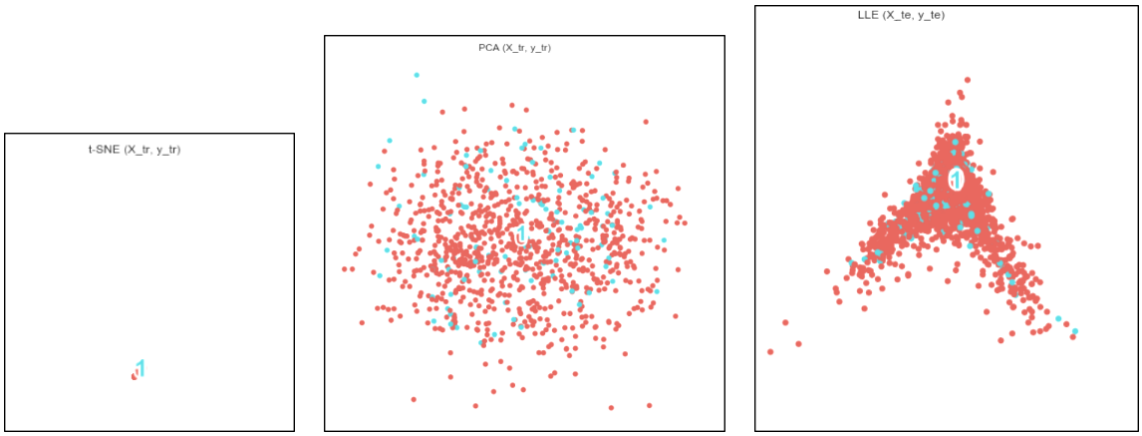

Figure 8- Dimensionally Reduction 2D Visualization

Observation: No clear separation of classes is identified

5. Data Preparation

The data were first normalized.

```
#(1) Drop the drop 'ID_code' column first
x = df.drop(['ID_code'], axis=1)

#(2) Normalize the data
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
```

And stratified sampling was applied to split the given set into training and test set.

```
#(3) Do stratified sampling on 'target' (80% training and 20% test)
split = StratifiedShuffleSplit(n_splits=1, test_size=TEST_SIZE, random_state=RANDOM_SEED)
for train_index, test_index in split.split(df_clean, df_clean["target"]):
    strat_train_set = df_clean.loc[train_index]
    strat_test_set = df_clean.loc[test_index]
```

```
#Setup training and test set
X_tr = strat_train_set.drop("target", axis=1)
y_tr = strat_train_set["target"].copy()

X_te = strat_test_set.drop("target", axis=1)
y_te = strat_test_set["target"].copy()
```

6. Methods

6.1 Basic Methods

6.1.1 KNN Classifier

Used as a baseline for more complex classifiers, it is simple enough with one single hyperparameter (number of neighbours).

Pros & Cons	Observations
Simple algorithm	N/A
No assumptions about data distribution (useful for nonlinear data)	N/A
High accuracy (relative) is pretty high but not competitive in comparison to better-supervised learning models	AUC score is ok but not as high as the rest of the models, could also be a consequence of stopped grid search at 300 neighbours due to time constraints
Computationally expensive	Takes a long time to train. Each training took around 1 hour and 30 minutes.
High memory requirement	The saved model is as big as a training set
Sensitive to irrelevant features and the scale of the data	No impact: Data was scaled as part of the pre-processing pipeline

6.1.2 Logistic Regressor & SGDClassifier

To use more complex classifiers, I selected generalized linear models and used the default parameters to evaluate them.

Pros & Cons	Observations
The result easily to interpret, and the output can be interpreted as a probability.	N/A
Good for cases where features are expected to be roughly linear, and the problem to be linearly separable.	N/A
High accuracy	AUC score is ok within the range of others
LR Fast Computation	Takes short time to train (10s)
SGD Fast Computation	Takes short time to train (3s)
LR Cannot handle categorical(binary) variables well	N/A
LR Use it if you expect to receive more training data in the future and want to quickly be incorporate into the model.	The model was offline with extensive amount of instances.
LR Can be used with kernel methods	We explored this during a deeper model exploration.

6.2 Ensemble

6.2.1 Random Forest Classifier

Pros & Cons	Observations
Train each tree independently, using a random sample of the data, so the trained model is more robust than a single decision tree and less likely to over-fit	N/A
Good for parallel or distributed computing.	N/A

Computationally expensive	Takes a long time to train depending on the number of estimators.
Good with uneven data sets with missing variables	The data set was cleaned before the feed. Unobserved result.
Calculates feature importance	Used (refer to Feature Importance section)
Train faster than SVMs	Tried to train SVC and gave up due to the time consumption.

6.2.2 Gradient Boosting Classifier

Pros & Cons	Observations
Build trees one at a time, each new tree corrects some errors made by the previous trees, the model becomes even more expressive.	N/A
Usually perform better than Random Forests, but are harder to get right. The hyper-parameters are harder to tune and more prone to overfitting. RFs can almost work "out of the box".	It outperformed the random forest classifier.
Training takes longer since trees are built sequentially	Training did not take as long as a random search in a random forest.

6.2.3 AdaBoost Classifier

Pros & Cons	Observations
Meta algorithm that combines weaker classifiers	N/A

Can be used with many different classifiers	We used the basic stump
Commonly used	N/A
Simple to implement	N/A

7. Feature Selection

After failed attempts to improve the performance, I attempt to analyze the features' importance using ExtraTreesClassifier.

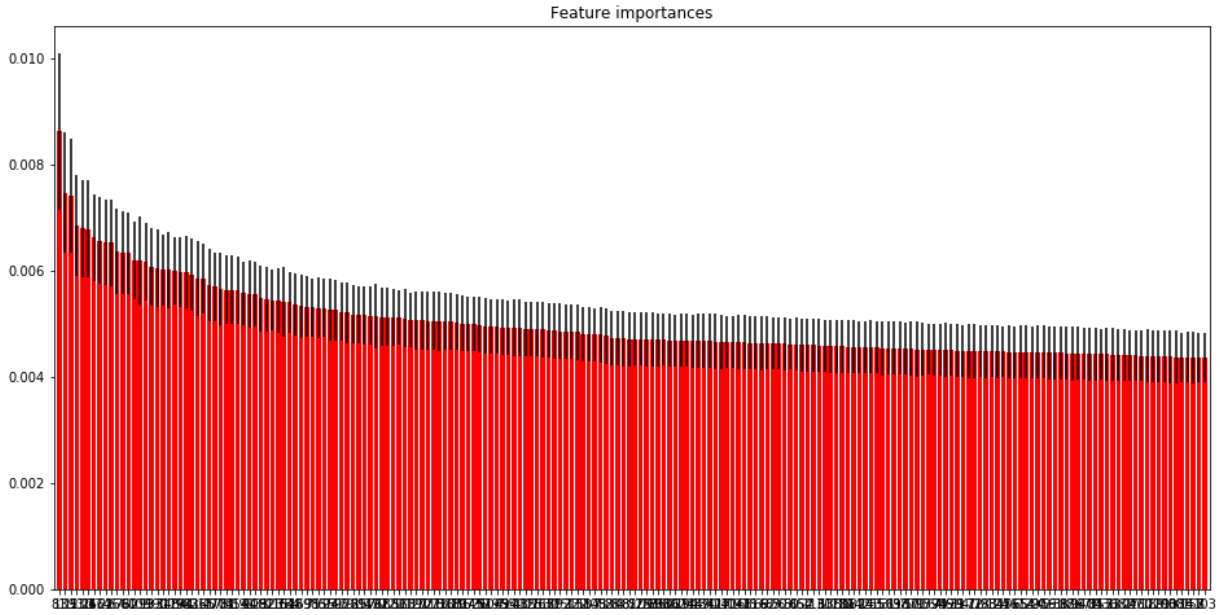

Figure 9 - Feature Importance

The ExtraTreesClassifier (n_estimators=1024) provides the 'feature_importances_' to rank how much influence each feature has on the training. The following shows a sample of the results and the ROC and AUC plots.

```
#Show the feature importance
importances = extraTreesClassifier.feature_importances_
std = np.std([tree.feature_importances_ for tree in extraTreesClassifier.estimators_], axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")
for f in range(X_tr.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
```

```
Feature ranking:
1. feature 81 (0.008625)
2. feature 139 (0.007473)
3. feature 12 (0.007412)
4. feature 53 (0.006859)
5. feature 110 (0.006794)
6. feature 26 (0.006787)
7. feature 146 (0.006630)
8. feature 174 (0.006565)
9. feature 22 (0.006530)
10. feature 6 (0.006525)
11. feature 166 (0.006362)
12. feature 76 (0.006340)
13. feature 80 (0.006329)
14. feature 2 (0.006204)
15. feature 109 (0.006183)
16. feature 99 (0.006176)
17. feature 133 (0.006066)
18. feature 190 (0.006046)
```

Figure 10 - Feature Importance Weights Sample

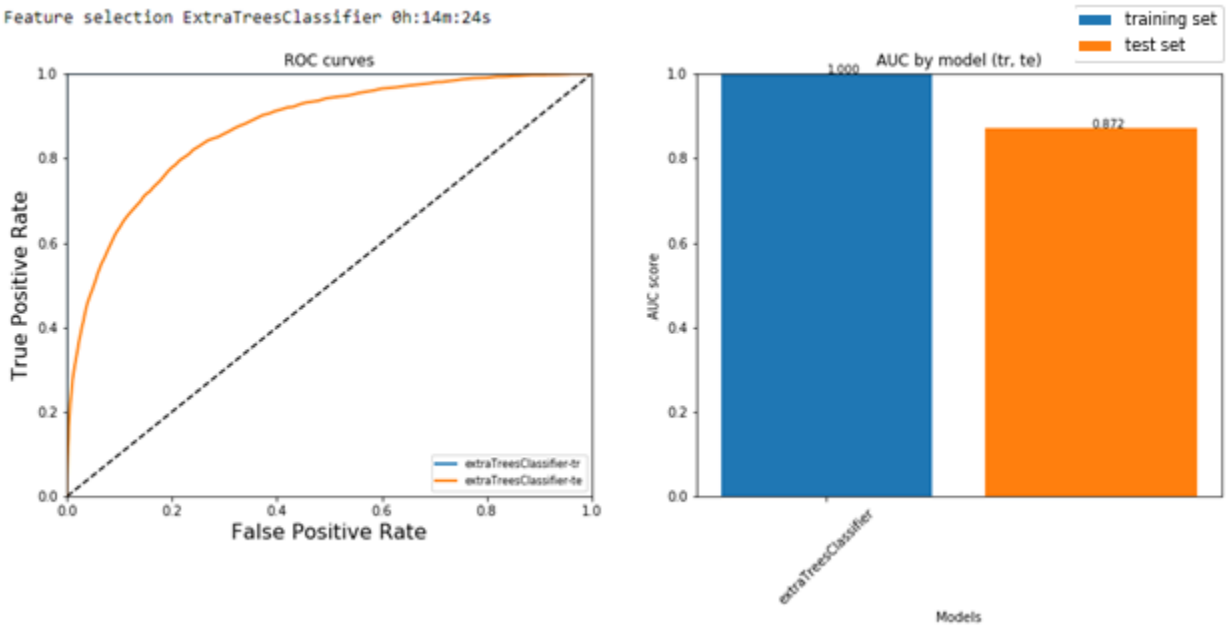

Figure 31 - ExtraTrees Classifier AUC

Arbitrarily reduced the dimensions to 50 features in order allow for enable running an SVC within a reasonable training time; I tested three different models with default parameters and compared their performances. The SVC gave good results and could be explored further but in the interest of this exercise, I decided not to dues to the lack of time.

```
#Filter by the 50 most important features
SELECTED_NUM_FEATURES=50
featureSelector = FeatureSelector(indices[:SELECTED_NUM_FEATURES])
X_tr_filtered= featureSelector.transform(X_tr)
```

Figure 12 - 50 Most Important Feature Selection

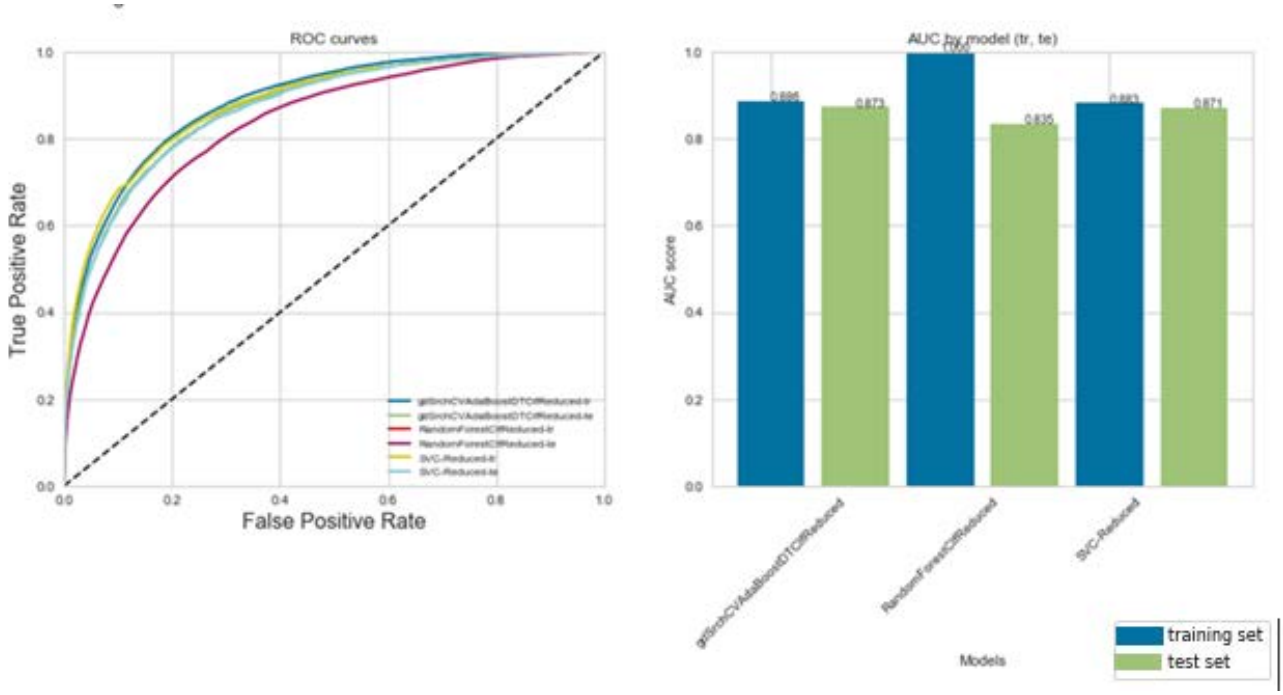

Figure 13 - AUC Comparison with Selected Features

8. Results & Comparison

8.1 Simpler Models

The following graph shows the results obtained from the Logistic Regressor with default parameters (dfltLogisticRegression), the SDG Classifier with default parameters (dfltSGDClf) as well and the best of the grid search conducted on the KNN Classifier (n_neighbors=300 and weights=distance).

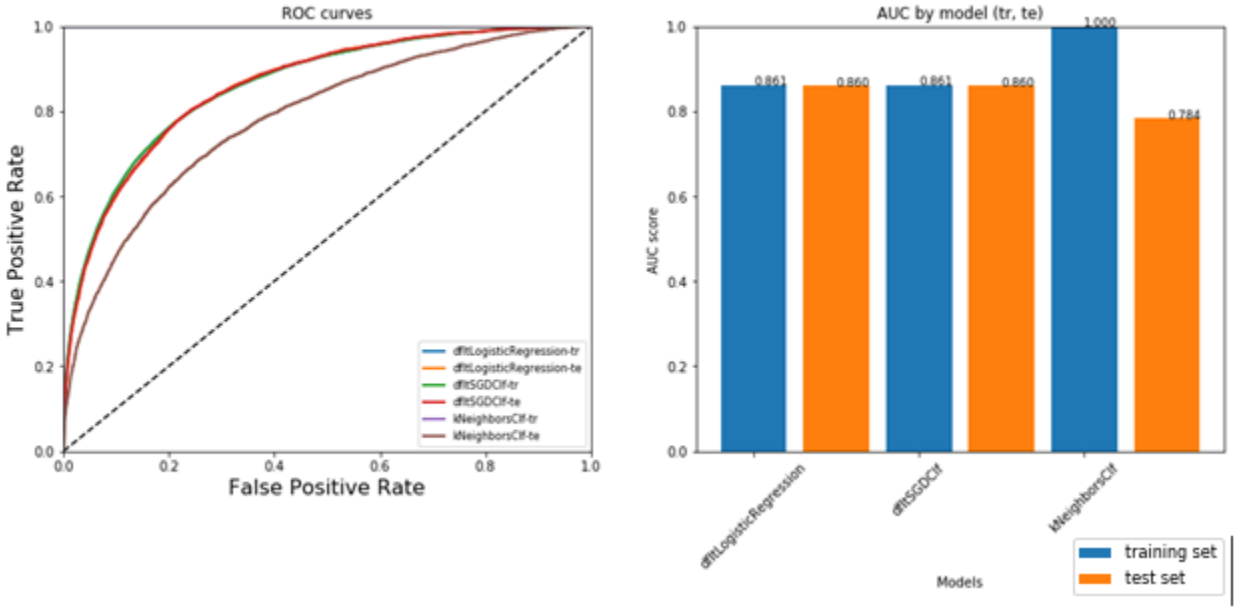

Figure 4 - Simpler Models AUC Comparison

8.1.1 Observation

The KNN Classifier took a considerable amount of time anywhere between 60 to 90 minutes to evaluate each model (grid search). Meanwhile, the Logistic Regressor trained in 10 seconds and the SDG Classifier took 3 seconds.

Note: Due to time constraints, the grid search on the KNN Classifier was stopped at 300 n_neighbors.

8.2 Ensemble

Three ensemble models were explored:

1. Random Forest Classifier
2. Gradient Boosting Classifier
3. AdaBoost Classifier

The following graph shows the results of each model (using default parameters).

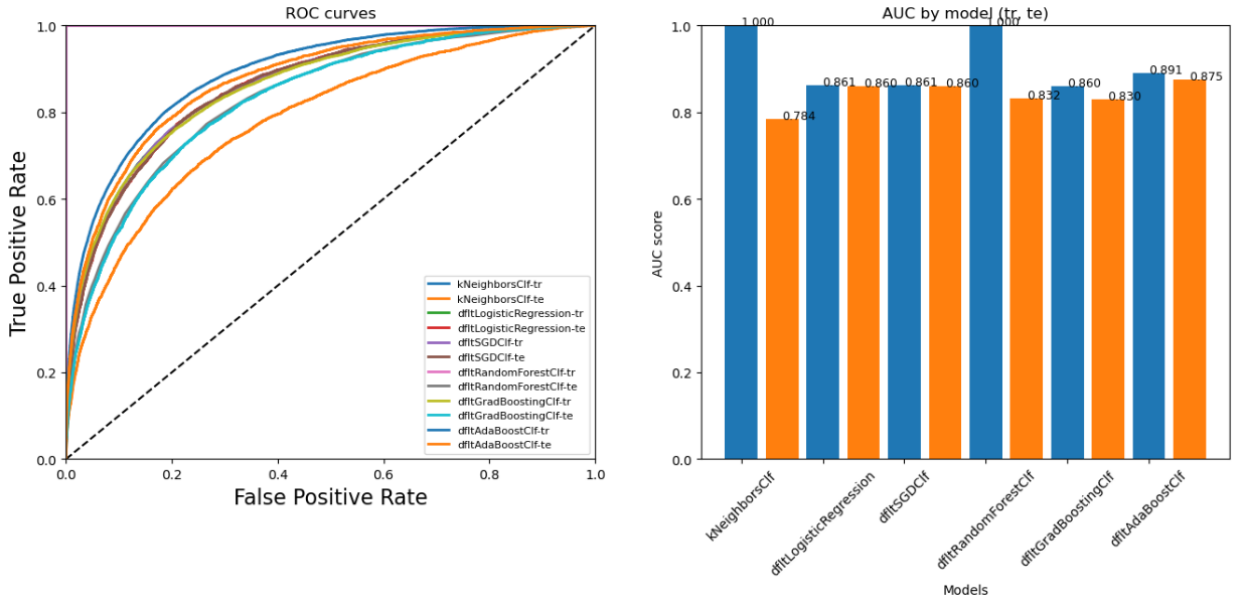

Figure 5 - Ensemble Models AUC Comparison

8.2.1 Observation

Out of the three Ada Boost Classifier gives the best score.

8.3 Hyper Parameter Tuning

Two models were selected for further analysis (hyperparameter exploration)

- Logistic Regressor
- AdaBoost Classifier
- Random Forest

The grid defined for each of the methods is as follow:

Model	**Grid Search**

Logistic Regressor	param_grid=[
	{
	"C":np.logspace(-3,3,20),
	"max_iter":[100, 200, 400],
	"penalty":["l1","l2"],
	"solver": ["liblinear", "saga"]
	},
	{
	"C":np.logspace(-3,3,20),
	"max_iter":[100, 200, 400],
	"penalty":["l2"],
	"solver": ["newton-cg", "lbfgs", "sag"]
	}
	]

AdaBoost Classifier	param_grid = {
	"base_estimator__max_depth": [1,2,3,4,5,6],
	"base_estimator__criterion": ["gini", "entropy"],
	"base_estimator__splitter" : ["best", "random"],
	"n_estimators": np.exp2(np.arange(1,10)).astype(int)
	}

Random Forest	I attempted to explore further the Random Search with Cross Validation for the Random Forest classifier. param_distributions = { 'bootstrap': [True, False], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None], 'max_features': ['auto', 'sqrt'], 'min_samples_leaf': [1, 2, 4, 10, 20], 'min_samples_split': [2, 5, 10, 20], 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 2000]

8.3.1 Best Parameters:

Logistic Regressor	Grid Search CV Logistic Regression 2h:9m:57s {'C': 0.07847599703514611, 'max_iter': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
AdaBoost Classifier	Grid Search CV ADA Boost Classifier 17h:30m:5s AdaBoostClassifier(algorithm='SAMME.R', base_estimator=DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=1, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=42, splitter='random'), learning_rate=1.0, n_estimators=512, random_state=None)
Random Forest	It ran for at least 72 hours. Since the results obtained do not improve the ones found by the AdaBoost, I decided to stop the search.

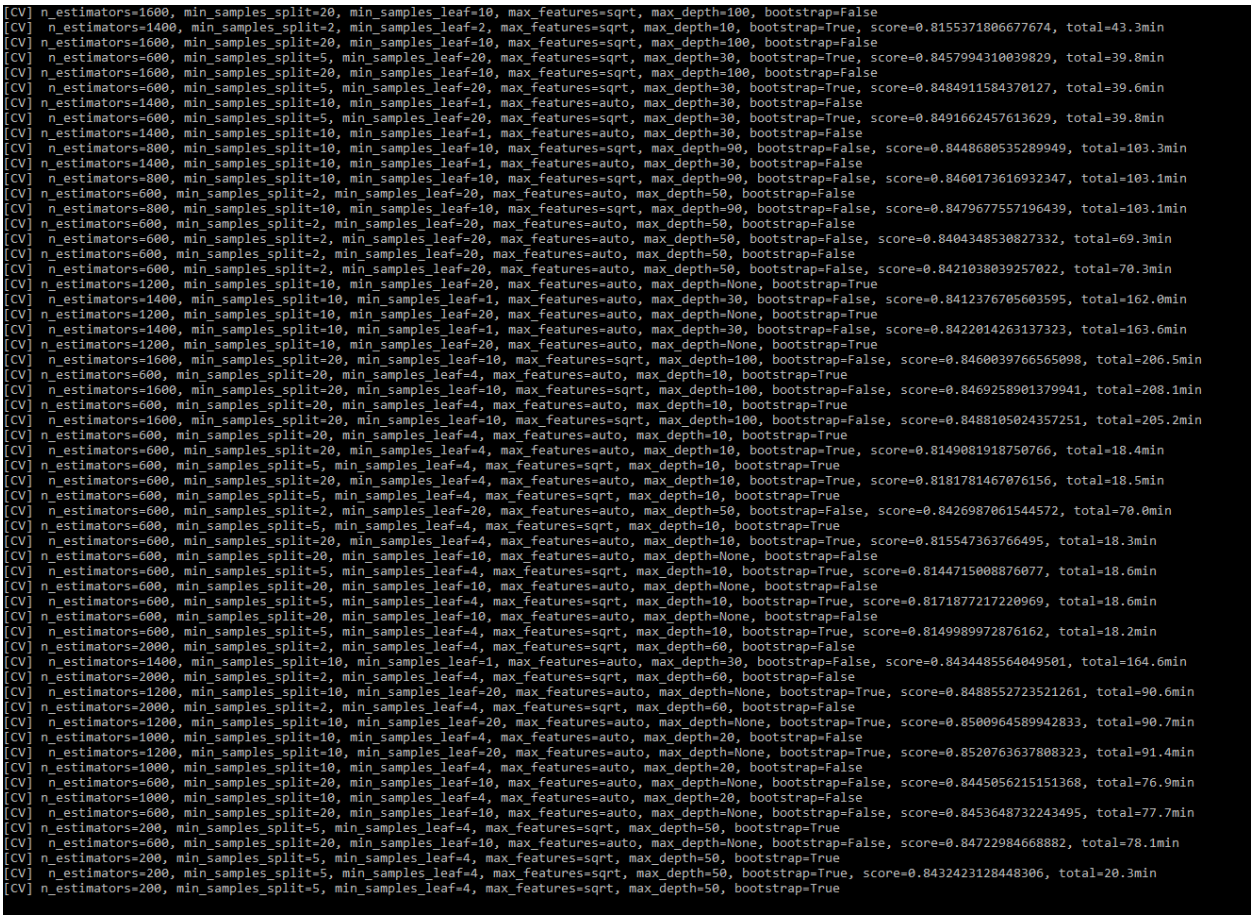

Figure 6- Random Forest Randomized Search

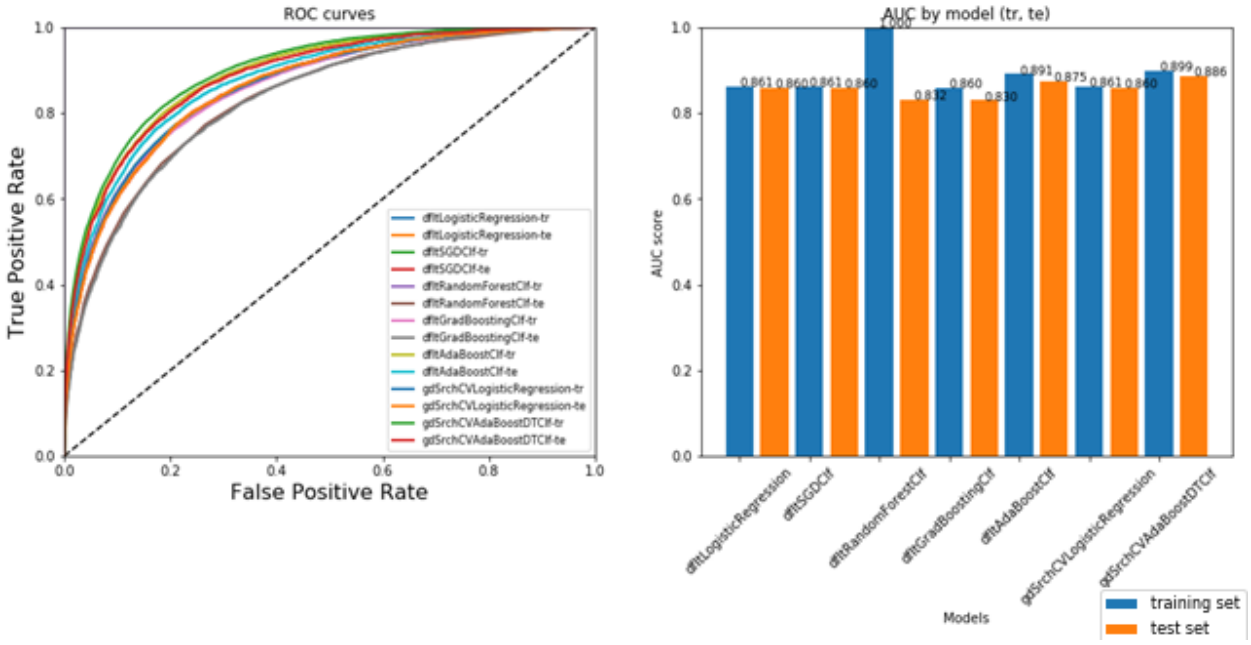

Figure 7 - Comparison after Hyper Parameter Tuning

Observations:
The results show no improvement on the Logistic Regressor (from the default parameters) but the AdaBoost Classifier improved from AUC 0.891 to 0.899 on the training set and 0.875 to 0.886 on the test set.

The best score is 0.899 on the training set and 0.886 on test set using AdaBoost Classifier.

9. Methods Performance Measure

Performance measurement was done on all base methods in the study. The performance measurements include accuracy, precision, recall, and F1-Score. These measurements are used to evaluate the performance of the methods in predicting the target variable.

The results are shown in the table below:

Method	KNeighbors Classifier	Logistic Regressor	SGD Classifier	Random Forest Classifier	Gradient Boosting Classifier	AdaBoost Classifier
Accuracy	89.93%	91.32%	90.97%	89.95%	90.19%	91.70%
Precision	83.71%	68.19%	79.42%	0.00%	85.93%	66.46%
Recall	89.93%	25.60%	13.73%	0.00%	2.89%	35.20%
F1 Score	85.21%	37.22%	23.41%	0.00%	5.58%	46.02%

Observation:
KNeighbors Classifier and Logistic Regressor appear to have the highest overall performance, with the highest accuracy, precision, and F1-Score. However, it is important to consider other factors such as computational complexity and model interpretability when selecting a method for a given task.

10. Challenges

Experience & Knowledge
This project provide me with an excellent learning experience, with real-life data; other Kaggle's competitors provided great material to read and expand my knowledge on the field with applicable examples.

Features Explainabilitye
Lack of feature labels makes this problem harder as no domain knowledge can be applied to help evaluate features' importance or relevance to outcome.

Computing power of effective algorithm
During the process, I experienced firsthand the challenges of doing hyperparameter tuning in some of the algorithms and the computing limitation even with this small set of data.

11. Future

Feature Engineering
Given the lack of domain knowledge is not easy to do feature engineering but it would be interesting to see if it makes any difference with the already tested models.

Further study new models
A lot of community members from Kaggle expressed good results with this decision tree boosting algorithm LightGBM.

Experience
Continue learning and gaining experience with real-life models.

12. References

Kaggle, (2019). Santander Customer Transaction Prediction. From https://www.kaggle.com/competitions/santander-customer-transaction-prediction

13. Github Link

https://github.com/leotheoilman/Santander-Customer-Transaction-Prediction