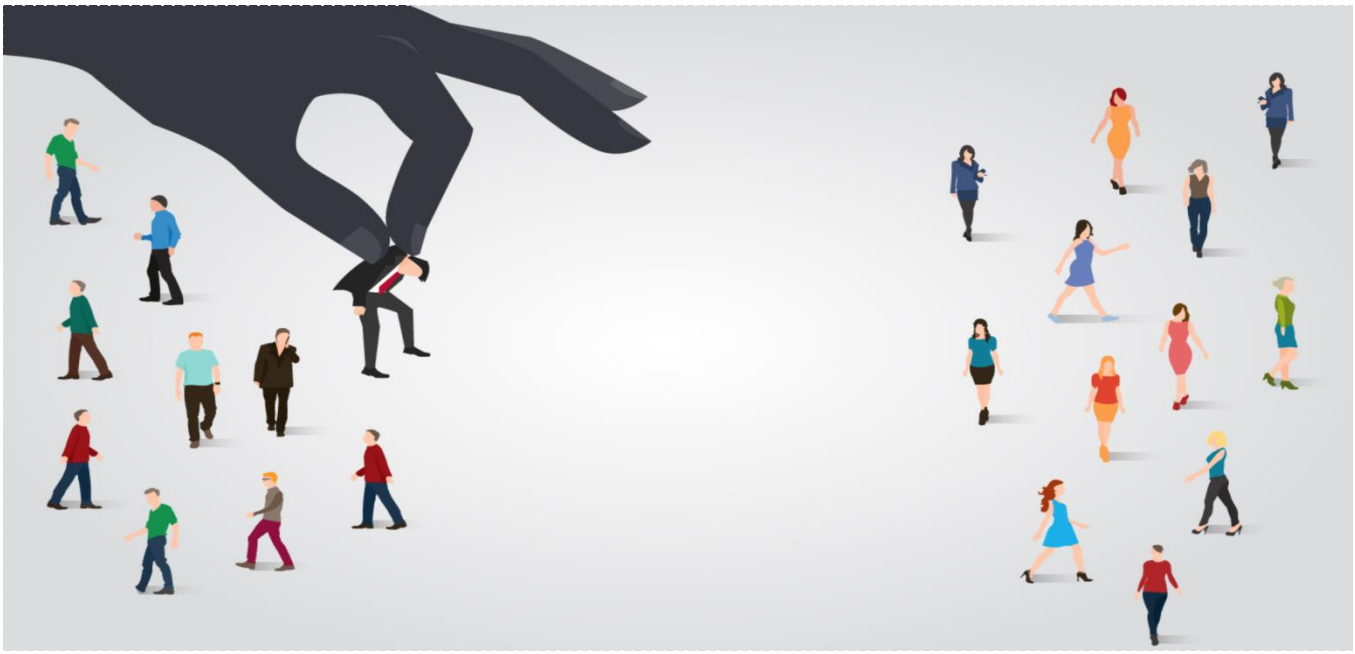
Univ.AI

SANTANDER CUSTOMER SATISFACTION



Report submitted as a part of ML-2 Project

Team Outliers

MANILA DEVARAJ	11699
KEERTHI JAYARAM	11688
NEHAL SHARMA	11675
GURUSHANKAR	500

ABSTRACT

In this report we describe our approach for building data models to predict which customers are unsatisfied. The raw dataset and the problem statement are taken from the Santander Customer Satisfaction competition on Kaggle. First, we introduce the problem statement and description of the dataset. Then we go on to do Exploratory Data Analysis on the data to understand the features and transform the data. After that we build a Baseline model against which we compare our Ensemble models. Finally we discuss the results obtained and draw relevant conclusions. The idea of this notebook is to provide an end-to-end approach to this challenge within the scope of ML-X course syllabus.

CONTENT

CHAPTER 1

INTRODUCTION

Problem Statement:

Santander, a Spanish bank has provided data related to their customers in an attempt to improve their customers' banking experience. If the bank can accurately determine its customer's satisfaction, it can improve its business by a large margin.

The Santander Customer Satisfaction competition was launched in 2016, with the aim of finding the best models to predict which customers are satisfied and which have complaints. The data set consists of 370 features and the target value we need to predict. The data provided is raw. We need to pre-process and clean it before applying the machine learning model.

The dataset is semi-anonymized, so it is unclear what a feature represents. The only clue we have is a header with a name for each feature that is clearly not randomly determined.

About the Dataset:

The data consists of multiple sets of customer features uniquely identified by an ID. The training dataset consists of 76020 observations with **371** observable features. Target column is the variable to predict. It equals one for unsatisfied customers and zero for satisfied customers. The test set consists of 75818 observations with no Target column, 370 observations and our task is to predict the probability that each customer in the test set is an unsatisfied customer.

It is very important to learn the features before applying cleaning methods and since the feature names were not clear, it was quite a challenge to predict which features would have an impact on our target variable.

CHAPTER 2

METHODOLOGY

✓ Exploratory Data Analysis and Feature Engineering

Feature Groups: The sheer number of features in this dataset makes it hard to individually analyze and discuss each feature, so instead similar groups have been identified in the dataset, assisted by their corresponding names. In Table 1, it is visible how the features with similar prefixes have been grouped.

Table 1: Variable Groupings

Substring	Example	Number (Raw)
'Num'	num_var37_med_ult2	87 (155)
'Ind'	ind_var13_0	46 (75)
'Saldo'	saldo_medio_var5_hace3	43 (71)
'Imp'	imp_op_var39_comer_ult1	21 (63)

Table 1

- We noticed that the data is highly imbalanced:
  1. Happy customers (0): 96.05 %
  2. Unhappy customers (1): 3.95 %

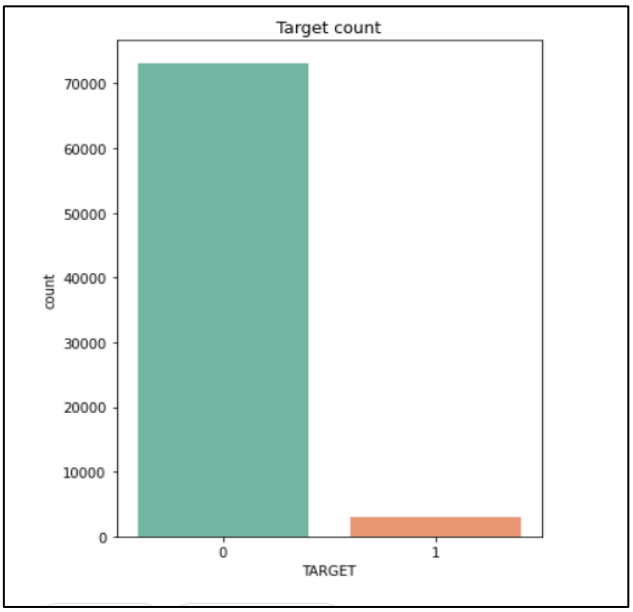


Figure 1

- We performed the following steps to combat this data imbalance:
    1. We checked and removed Zero variance features
    2. We checked for and removed features that were duplicated (i.e., they shared same values across data points)
    3. We removed sparse features (features that have little information that is, if a feature has 99 percentile value to be 0 then it is considered as a sparse feature)
    4. We removed correlated features that had low correlation with target and had high correlation with each other (keeping only one of them)
    5. After performing the above steps, the number of our columns reduced from 370 to 93.

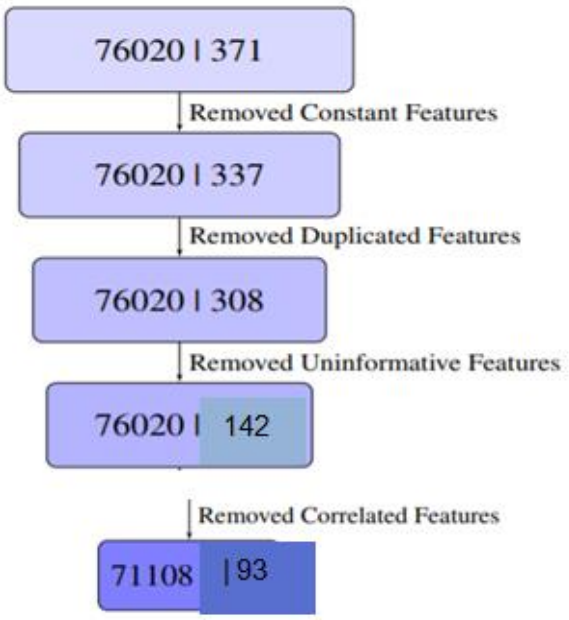


Figure 2

- Balancing the data using SMOTE:
    1. Out of the many ways to deal with imbalanced data, we are going to use SMOTE or Synthetic Minority Oversampling Technique to up sample the minority class.
    2. SMOTE basically uses a nearest neighbors algorithm to generate new and synthetic data we can use for training our model.
    3. After performing the above steps, we observed equal distribution in our dataset (50% each).

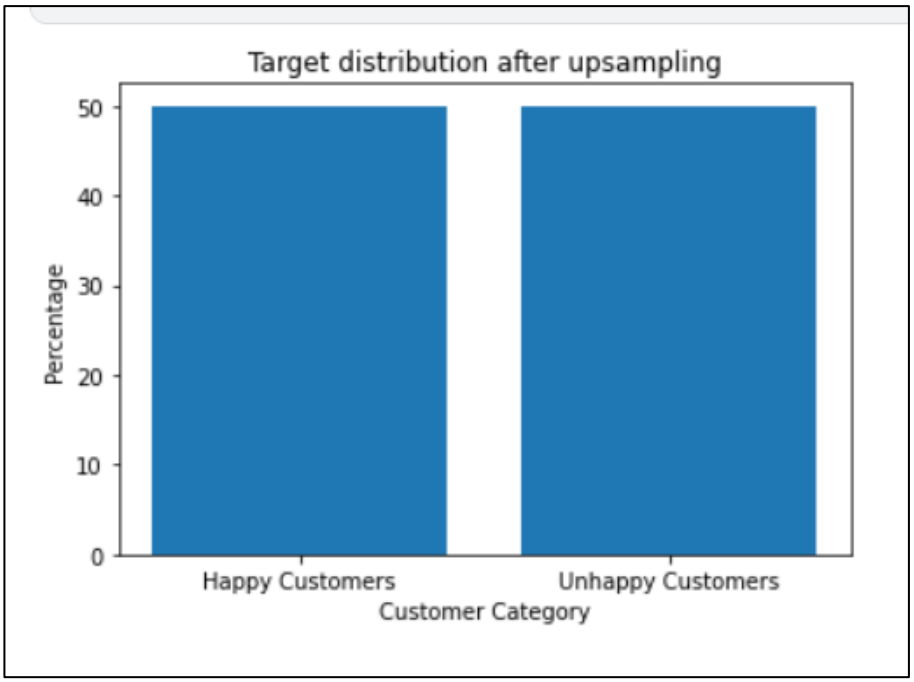


Figure 3

✓ Modeling

The performance metric we have considered is AUC. This was a binary classification problem i.e, either classify that the customer is satisfied with the services or not satisfied. Customers who are unsatisfied are a huge part of the retention program.
The retention program cost $10 for each predicted unsatisfied customer and a customer lifetime value of $1000 if we lose customers.
In the classification task we can have the following scenarios:

1. ***False Positive (FP):*** Classify the customer as UNSATISFIED but he is SATISFIED. Cost: 10, Earn: 0;
2. ***False Negative (FN):*** Classify the customer as SATISFIED but he is UNSATISFIED. Cost: 0, Earn: 0;
3. ***True Positive (TP):*** Classify the customer as UNSATISFIED and he is UNSATISFIED. Cost: 10, Earn: 100;
4. ***True Negative (TN):*** Classify the customer as SATISFIED and he is SATISFIED. Cost: 0, Earn: 0;

Baseline models:

1. ALL customers are unsatisfied and we send to EVERYONE.
   (**Accuracy:** 3.96%)
2. NO customer is unsatisfied and we send to NO ONE.
   (**Accuracy:** 96.04%)
3. **Logistic Regression:** It is a relatively 'simple' machine learning algorithm and we expect fast, but not great results. It tries to attach the best constant values to how features interact with the target, based on the train set, minimizing an error term. It then applies this same formula to the test data set. It is pursued here as a baseline in order to compare to more sophisticated models.
   (**Hyper parameters-** alpha:[1000, 100, 10, 1, 0.1, 0.01, 0.001])
   (**Train Accuracy:** 83.5% | **Test Accuracy:** 80.1%)

Classification report for Logistic regression model:

```
print(classification_report(ytest, best.predict(Xtest)))
```

	precision	recall	f1-score	support
0	0.98	0.81	0.89	21900
1	0.11	0.59	0.19	906
accuracy			0.80	22806
macro avg	0.55	0.70	0.54	22806
weighted avg	0.94	0.80	0.86	22806

Table 2

Ensemble models:

1. **Random Forest:** The Random Forest model generates multiple decision trees. Only a subset of predictive features is now considered during each split that is randomly selected. The decision trees lead to one single prediction together by averaging all the predictions they give individually. The parameters of a Random Forest model are really important and need to be tuned appropriately. N_ESTIMATORS determines the number of trees in the forest. MAX_FEATURES controls the maximum random amount of features to consider when determining a best split during the algorithm. MAX_DEPTH limits the depth of a tree in the random forest. N_JOBS controls the number of processors. Trees in a random forest can be made in parallel, so the more cores working, the less computation time needed.

   (**Hyper parameters-** max_depth: [3, 4, 5], max_features: range(2,10,1)

   BEST FIT:

```
print(RF_model.best_params_, RF_model.best_score_)
```

```
{'max_depth': 5, 'max_features': 9} 0.9126508537274528
```

   (**Train Accuracy:** 81.3% | **Test Accuracy:** 78.5%)

Classification Report for RF model

```
print(classification_report(ytest, RF_model.predict(Xtest)))
```

	precision	recall	f1-score	support
0	0.98	0.79	0.88	21900
1	0.11	0.61	0.18	906
accuracy			0.79	22806
macro avg	0.54	0.70	0.53	22806
weighted avg	0.95	0.79	0.85	22806

Table 3

We observed that although the overall accuracy of the model is less than that of Logistic Regression, the recall score for the Class:1 (unsatisfied customers) is much better than the LR model.

Hence, the Random Forest model is doing a better job at classifying dissatisfied customers.

2. **XGBoost:** It is a method where the outcome is formed by a combination of multiple trees. The trees are built iteratively. Every time a new tree is built it focuses on parts where the previous ones make mistakes, by assigning higher weights to these instances. This stands in *contrast* to Random Forest, where trees are made independently from each other. The parameters are again of grave importance here. N_ESTIMATORS and MAX_DEPTH are the same as with the Random Forest model. LEARNING_RATE is a parameter that controls the speed and preciseness of the model. The lower it is, the more accurate your model becomes, but the more rounds it needs to converge. It represents a constant times how much the next tree built affects current predictions. XGBoost directly avoids over fitting by promoting simplicity of models in the objective via regularization, unlike Random Forest that only limits the way trees can grow by imposing restraints.

$$\min Obj(\Omega) = \text{Training Loss Function} + \text{Regularization}$$

$$\min Obj(\Omega) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \alpha \sum_{i=1}^{k} |w_i| + \lambda \sum_{i=1}^{k} w_i^2 + \gamma T$$

REG_ALPHA is a parameter for l1 regularization. LAMBDA is a parameter for l2 regularization. GAMMA is a regularization parameter that multiplies itself with the number of leaves. This regularization is in place to penalize the objective for building overtly complex trees to avoid over fitting.

(**Hyper parameters-** n_estimators: range(10, 20),
Learning rate:[.05,.06,.07,.08,.09,.1,0.15,0.2,0.25,0.3]
Best hyper parameters- learning_rate: 0.3, n_estimators: 19
Best AUC value: 0.9545920961173534
(**Train Accuracy:** 95.5% | **Test Accuracy:** 81.7%)

Classification Report for XGBoost model

```
print(classification_report(ytest,xgb_model.predict(Xtest)))
```

	precision	recall	f1-score	support
0	0.98	0.88	0.93	21900
1	0.16	0.55	0.25	906
accuracy			0.87	22806
macro avg	0.57	0.72	0.59	22806
weighted avg	0.95	0.87	0.90	22806

Table 4

CHAPTER 3

INTERPRETATION OF THE MODELS

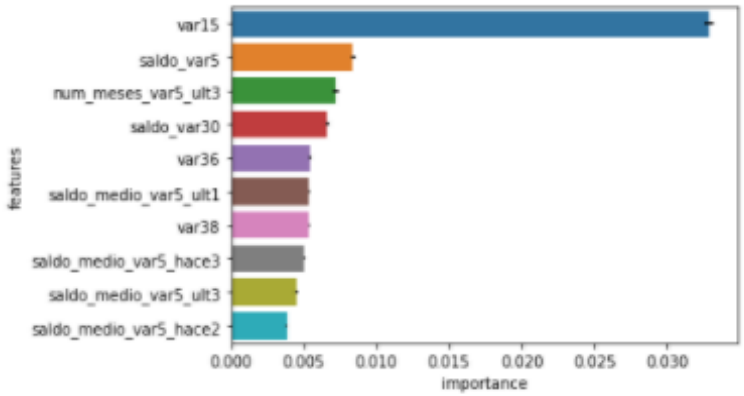
3.1 PERMUTATION IMPORTANCE

The permutation importance function calculates the feature importance of estimators for a given dataset.

Feature importance for 'RF_model' on training dataset

```
RF_df=p_importance(RF_model,list(Xtrain.columns),RF_pi['importances_mean'],RF_pi['importances_std'])
sns.barplot(data= RF_df, y='features',x='importance', xerr=RF_df.importance_std)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f20b6027650>
```



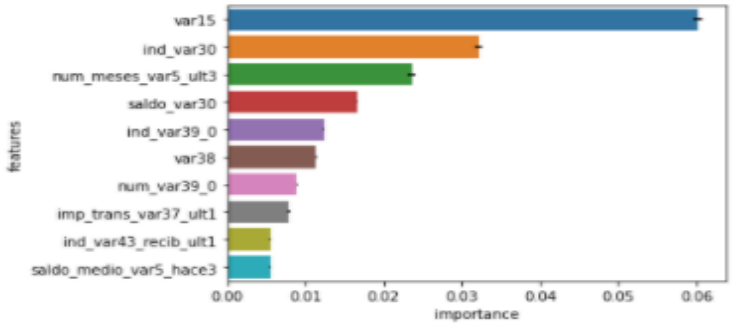
Feature importance for 'XGBoost' on training dataset

```
#Calculate permutation importance and plot a bar chart
pfi = permutation_importance(xgb_model, Xtrain, ytrain)
```

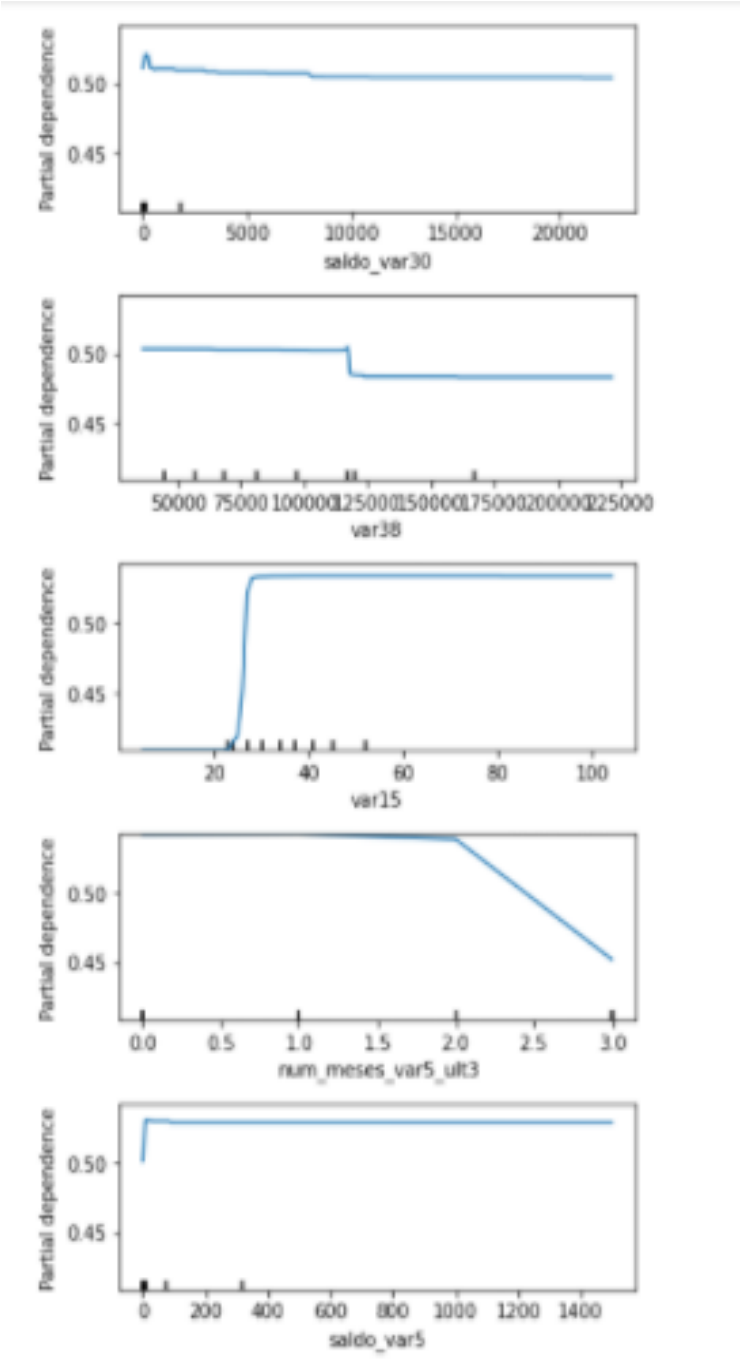
```
pfidf = p_importance(xgb_model, Xtrain.columns, pfi['importances_mean'], pfi['importances_std'])

#Plot the bar plot
sns.barplot(data= pfidf, y='features',x='importance', xerr=pfidf.importance_std)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f20b59cbc50>
```



3.2 PARTIAL DEPENDENCE PLOTS



CHAPTER 4
CONCLUSION AND FUTURE SCOPE

4.1 CONCLUSION

In this project, we did a study on how to preemptively understand if customers of Santander will be dissatisfied using Machine Learning. An anonymized dataset, to protect the privacy of the customers, led to difficulties in asserting what could be relevant or not, especially in light of a “huge feature set”. However, a thorough data analysis discerned the meaning and interpretation of several features. A Python implementation utilized the Logistic Regression, Random Forest and XGBoost algorithms, carefully tuned, in order to predict if the customer unsatisfied or not.

4.2 FUTURE SCOPE

The project can be extended to do better by implementing Deep Learning models or a combination of optimal models to enhance performance. We can use Bayesian optimization to find the best hyper parameters for each of the existing models. For further iterations on this project, we could work on creating new features if need be and spend more time breaking down the semi-anonymized features for thorough analysis.

CHAPTER 5

REFERENCES

1. *https://www.kaggle.com/saga21/customer-satisfaction-models-explainability*
2. *https://medium.com/analytics-vidhya/how-to-solve-a-machine-learning-problem-example-with-code-695e623102c8*
3. *https://beta.vu.nl/nl/Images/werkstuk-elsen_tcm235-865964.pdf*
4. *https://towardsdatascience.com/santander-customer-satisfaction-a-self-case-study-using-python-5776d3f8b060*
5. *https://github.com/univai-ml2-c1/Lab4*
6. *https://www.kaggle.com/cast42/exploring-features*