# **Final Report**

**Santander product Recommendation**

Introduction

The purpose of this project is to build a better recommendation system and presumably do a better job of advertising services to the people who are likely to use them, not only saving Santander money, but also providing a better customer experience over all. As this is a bank, the products of the bank are Credit Cards, Debit Cards, Payroll Services, electronic banking, tax services, etc.

Data Source & Format

The data has been taken from the internet from the below URL.

<https://www.kaggle.com/c/santander-product-recommendation/data>

Below is a summary of the dataset size & my system configuration.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset** | **Size** | **OS** | **RAM** |
| Training | 2.3 GB | windows | 8 GB |
| Testing | 107 MB |  |  |

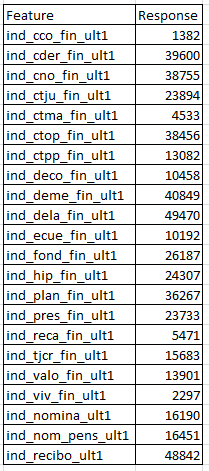
After downloading the data, I started with data pre-processing. Using ‘R’ application I read 5million rows and started preprocessing of data. I used Stratified Sampling for splitting data set into Train and test. I used this sampling because it divides the dataset on strata and helps to overcome bias if the dataset is not balanced.

The training set is a combination of Numerical & Categorical attributes. Also, most of the columns have null values. The training data has 30 features with 17 classes. Attached is an excel containing information about the data dictionary.



The variables in the format ind\_\*\_fin\_ult1 are the response variables, and these represent the products that are offering by the bank. We have 24 target variables and out of these 24 we are using only 22 features for modeling because of rare events in other two features. As the target variable, has more than two output values we need to use the multinomial distribution.

Table below displays the target variable and response.



We have data from May 2015 till June 2016 and we need to predict the products which customers are willing to purchase in June 2016.

Programming Languages & Packages

I used Python for initial pre-processing step. Later I replicated the same steps in ‘R’ and continued modeling in ‘R’. please find the below tables for the package list which I have used.

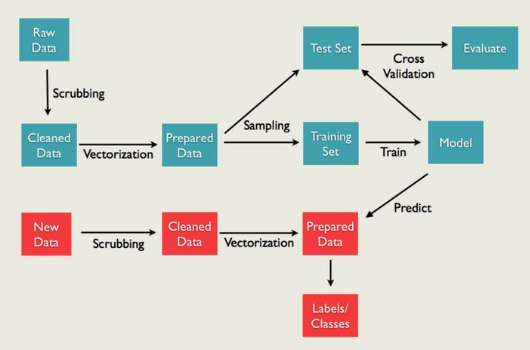
Python Packages:

|  |  |
| --- | --- |
| **Package** | **Purpose** |
| Pandas | Data Frames |
| Numpy | Data manipulation |
| Bokeh | Interactive visualisation |
| Sklearn | Machine Learning |

R Packages:

|  |  |
| --- | --- |
| **Package** | **Purpose** |
| Data.table | Data tables |
| dplyr | Data manipulation |
| ggplot | Visualisation |
| lubridate | Dates |
| Random Forest | Machine Learning |
| XgBoost | Machine Learning |

Below is the methodology that would be used for this project



Scoring Metric

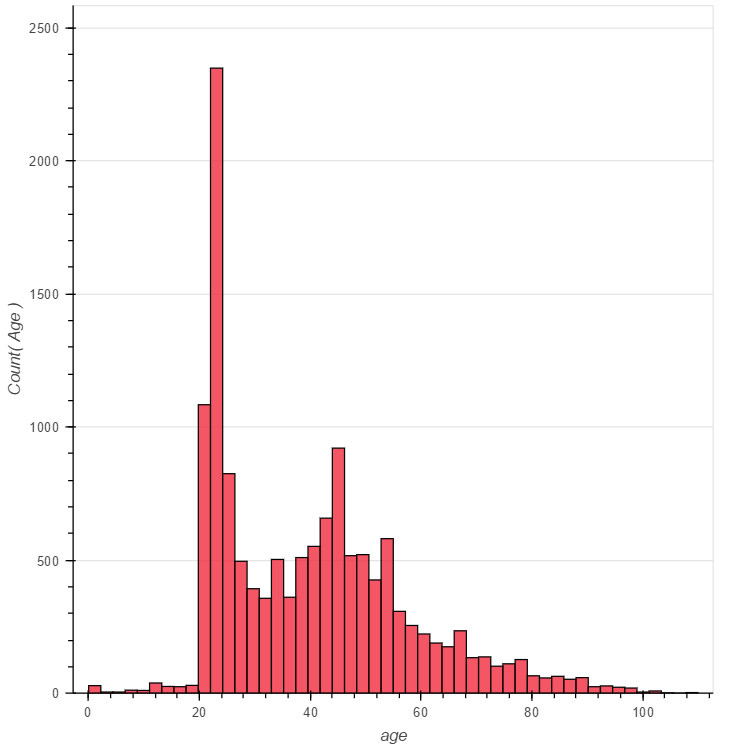
The scoring was evaluated using Mean Average Precision at 7 (MAP@7). The intuition behind this scoring metric is that it rewards solutions where the person actually added one of the items you recommended, and you get more points if the purchased item was earlier in your list of recommendations. You don’t lose any points for recommending products to people who don’t but anything. Therefore, we should recommend exactly 7 products to each customer, and place the most likely ones earlier in your list.

At high level, the basic strategy for this kind of problem is the following. For each product, predict the probability that each customer will own it in the following month. These predictions can be the result of the machine learning models, market basket analysis, etc. A list of recommended products can then be obtained by sorting these from most-likely to least-likely and removing the ones that are currently owned.

Solution Summary

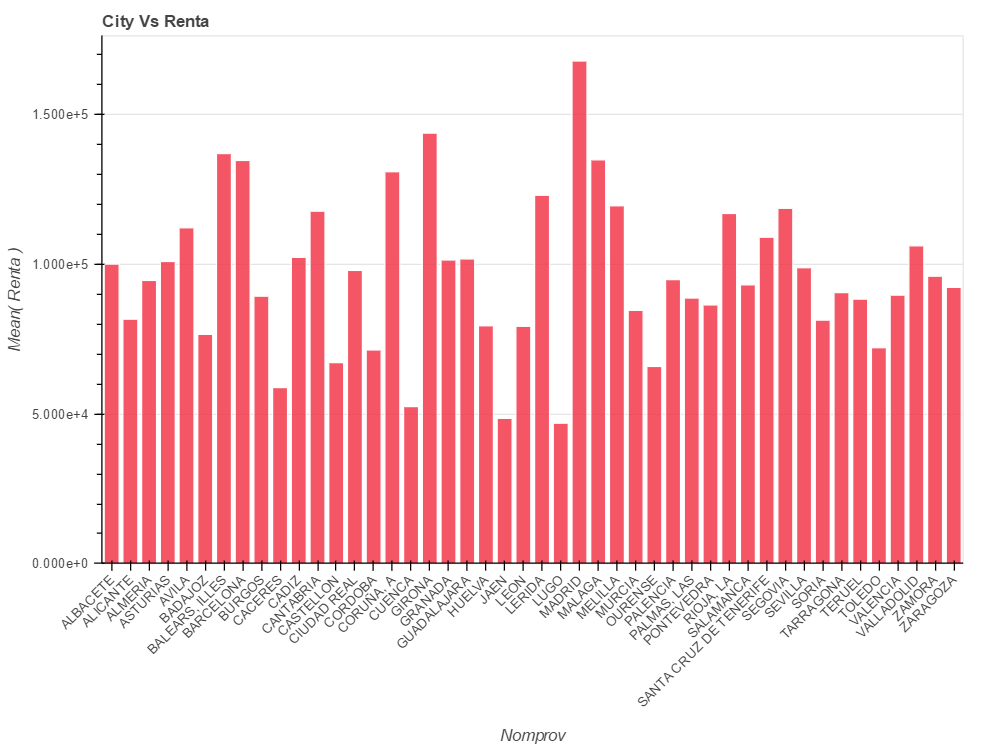
After identifying the scoring metric, I started with the Data Preprocessing step. Most of the columns in the dataset has NA values. I replaced the NA values for each of the column by finding the distribution of each variable. I used histogram and boxplot functions in ‘R’ for plotting the variables.

I started with ‘age’ predictor. Please find the below plot for distribution.



From the plot, it infers that it is bimodal distribution. So, I have divided the age into intervals and imputed missing values with the median of that interval. It doesn’t make sense that people with 2 years of age have bank accounts. So, I replaced the values with the median of the closest distribution.

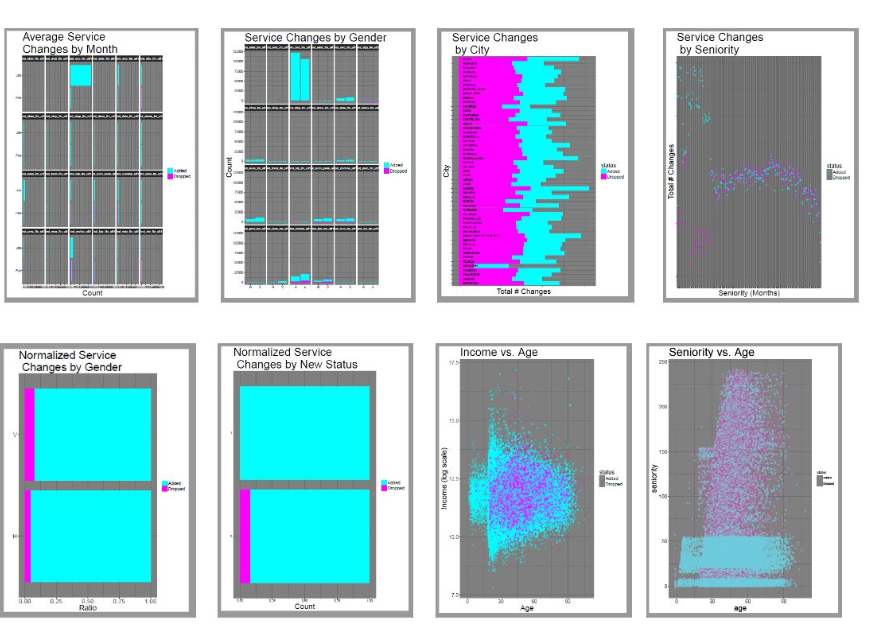
For imputing missing values of ‘Renta’ predictor, I have plotted the income for each city. Please find the distribution plot below:



We can see that income distribution by city is varying a lot. So, it makes more sense that for missing values we assign the median of the city’s income as the missing income.

For remaining categorical variables, I imputed the missing values with the mode of that variable.

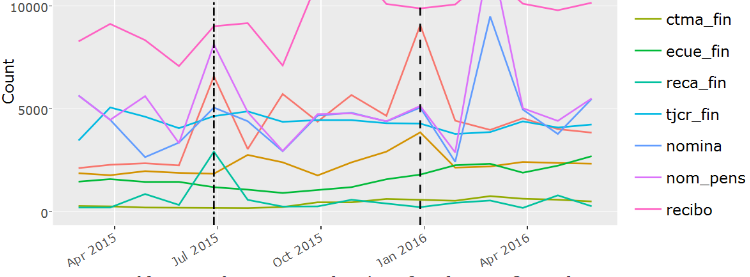
Apart from these, I have done few visualizations for understanding each variable in dataset.



After the Data Pre-processing is done, I started the Feature Engineering step which is very important in our project. I have done the Data Preprocessing step on my own and shared results with the entire team. I did pre-processing steps in Python as well as R.

Feature Engineering

As we have 13 months of data, I did forecasting to better understand the trend for each product. Please find the plot below for observing the trend of each product w.r.t year.



Clearly, we can say that there is seasonality over time. In particular, looking at the April data it can be seen that the product follows same trend over time so we are thinking of only considering ‘June 2015’ data for final predictions of the June ’16 data.

As the test data doesn’t have information for June 2016, I merged the train and test data by Customer id and updated the products that have been purchased by the previous months. For imputing the missing values after merging the two datasets, I followed the same approach which I have mentioned above. Below mentioned are the features which we have engineered. I did the numerical features part.

Engineered Categorical features:

1. ownership – binary feature indicating whether or not each product was owned 1-11 months ago (242 total features)
2. owned.within - binary feature indicating whether or not each product was owned within 1-11 months ago (242 total features)
3. segmento.change – indicates whether the value of segmento is different from last month than this month
4. activity.index.change – indicates whether the value of ind\_actividad\_cliente is different from last month than this month
5. month - what is the current month?

Engineered numerical features:

1. purchase.frequency - number of times each product has been purchased (22 features)
2. total\_products - total number of products owned 1-11 months ago (11 features)
3. months.since.owned - number of months since each product was last owned (22 features)
4. num.purchases - total number of products added 1-11 months ago (11 features)
5. num.transactions - total number of transactions 1-11 months ago (11 features). A transaction is defined as adding or dropping a product

I started Feature Engineering by adding a new column for the dataset for finding the customer status i.e Maintained (0), Dropped (-1), Added (+1). Later, I found the list of purchased products for each customer. This can be done by connecting each row to the corresponding one from the previous month. If the difference in the product’s ownership status is positive I appended the product to the list.

Next, I found product purchase frequency and number of transactions by month. Product Purchase frequency is a feature indicating number of times customer has added that product. I tried normalizing the purchase frequency by the number of months that the customer has been a member. I did this my assigning the index and then dividing the same index.

Later, I wrote another function for finding the Number of months a product was last owned. Suppose account X owns product Y for months 1-5 and then drops it, and then for an observation in month 12 we wish to know how many months since Y was owned, searching 11 months total. 11 months ago was month 1, and the product was owned, so we record a 11. That continues until we look 7 months ago at month 5, and record 7 months for the feature value. When we look 6 months ago at month 6, the product was not owned, and we don't do anything. The final value of the feature will be 7, the number of months since X owned Y.

After feature engineering is done, I started with modeling part. I tried two algorithms for the resulting dataset.

Random Forest

Random Forest is a bagging algorithm and will build forest of trees by randomly selecting the features. This is a collective learning method for classification and regression tasks, that operate by constructing a multitude of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time and outputting the class that is the [mode](https://en.wikipedia.org/wiki/Mode_(statistics)) of the classes (classification) or mean prediction (regression) of the individual trees. We can also find variable importance using this algorithm. Also, banking products usually are based on a lot of if-else conditions so decision trees can make more intuitive sense compared to other models. As Random Forest will avoid the over-fitting, with better accuracy when compared to other algorithms for this problem.

After discussing with professor, I took the 3 months data for training and tried to predict on preceding month. For training data I built only 20 trees. When I am increasing the number of trees I am somehow facing memory issues. I have added few interesting facts which I have learnt while running the Random Forest algorithm.

We have so many categorical variables in our dataset and in-built function of Random Forest in ‘R’ is considering only the categorical variables with 53 levels. If we have more levels then it will throw an error. So, I have two categorical variables in dataset which is violating the rule. I have removed those two predictors while modeling. Below are the results:

|  |
| --- |
| **Results** |
| Accuracy: 99.99% |
| Precision: 1% |
| Recall: 0% |
| Number of Trees: 20 |

From the above results, we can infer that tree model is highly imbalanced because the Precision and Recall scores are very less. After report-3 I tried to build 50 trees but the precision and recall scores are not improved. So, Random Forest will not provide good prediction results for this problem.

So, as a team we came up with another modeling technique which is called XGBoost algorithm.

XGBoost

XGBoost is a Boosting algorithm. As a team, we think this is best model for this problem because of additive modeling concept it uses. It is fast, can be run in parallel and supports the regularization L1/L2 (which would be useful for generalization purposes). Gradient boosting is a machine learning strategy where you build many simple models that are called weak learners. Commonly these are decision trees, but the idea can be generalized to essentially any model. Each model you build focuses on the mistakes of the previous one(s), which is achieved by reweighting the input data to the i’th weak learner based upon some error metric between the first i-1 models and the data. The output of the model as a whole is then a combination of the predictions made by each of the many weak learners.

There were two primary modeling strategies:

1. Repeated single-class classification: A separate XGBoost model was trained for each product using objective = "binary:logistic", and the target value was whether or not that product was owned, regardless of it being newly added or not. The result is that each set of predicted probabilities is the result of outputs from many models. The same input data was used for each of the sub-models. This method produced the best MAP@7 with a single model.
2. Multiclass classification: A single XGBoost model was trained using where the target variable was the product that was added. Using objective = "multi:softprob", probabilities for each class were obtained all at once. For customers that added multiple products, a single one was chosen at random as the target.

First model will use only June 2015 data and predict the June 2016. For the first model, we got map score of 0.30107.

The second model uses all the data. For the second model, we have done the Feature Engineering again. We added lag variables for each of the feature. The motivation behind including it is to give the model a prior knowledge of purchase behavior of the customer every month.

Lag Features

This is also important part in our project. For each entry, it was beneficial to consider not only the value of a feature for the current month, but also the value for previous months. For implementing this, I joined the data by account id, ncodpers, and to match the month with the lag month. For example, to add a 2-month lag feature to an observation in month 5, we want to extract the value of feature.name at month 3. I created lag features for ‘ind\_actividad\_cliente’, the customer activity index. For few percent of customers, I noticed that ‘ind\_actividad\_cliente’ is perfectly correlated with one of the few products ind\_tjcr\_fin\_ult1 (credit card), ind\_cco\_fin\_ult1 (current account), and ind\_recibo\_ult1 (debit cards).

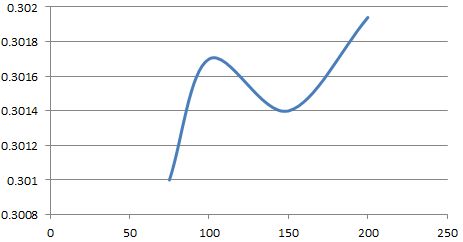
After adding the lag features to dataset, I implemented the XGBoost on entire dataset. Please find the results below:

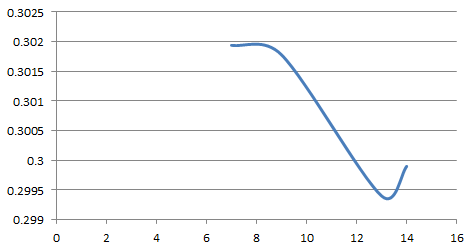
|  |  |
| --- | --- |
| Num Tree | MAP@7 |
| 75 | 0.301 |
| 100 | 0.3017 |
| 150 | 0.3014 |
| 200 | 0.30194 |

|  |  |
| --- | --- |
| Num Levels | MAP@7 |
| 7 | 0.30194 |
| 9 | 0.30178 |
| 13 | 0.2994 |
| 14 | 0.2999 |

First table briefly describes the trees and score achieved. The second table is about the depth of tree and score metric.

We plotted the number of trees vs score metric and depth vs scoring metric. From the graph it infers that, if total trees = 200 the scoring metric is high.





As the depth of tree increases, it is difficult for the observation to fit into the tree. So, the scoring metric score will decrease.

The best results have a MAP of 0.35. We are quite convinced since our model was explaining 90% of that variation.

Conclusion

* I would like to conclude that modeling results indicated that even with less data we have a decent model which means that we do not need a complex model always for getting better statistics.
* As feature engineering is iterative process in our project, we were able to improve the MAP score by better feature engineering and more data which means that it is always better to make a better model. There is slight improvement in the score if we use entire data for modeling. For this problem, slight improvement is also a good thing since it is better to spend more money on a better advertisement campaign than sending bad advertisements.
* I realized feature engineering and understanding data are very crucial for solving any Machine learning problems.
* We think that our approach is correct in solving the problem since the iterative process and feature engineering really helped and this is what made our model to perform better
* From the random forest results we observed that tree is highly imbalanced. To overcome the less resource or skewness problems, we identified an approach known as down-sampling. But, we didn’t try this in our project as XGBoost is giving better performance. SMOTE is good method for solving the imbalance problem. As we used MAP as our evaluation metric and we needed to recommend atleast 7 products and MAP weights the results even though we recommend our product at the end (but with less weight) but does not give a negative score if we don’t recommend a product.