Product Recommendation for Santander Bank Using Machine Learning Techniques

Adwoa Brako[1], Aira Domingo[2], Renzo Castagnino[3]

Columbian College of Arts & Sciences, The George Washington University, Washington, D.C.

[1]abrako64@gwu.edu

[2]airadomingo@gwu.edu

[3]rcastagnino@gwu.edu

Abstract

Santander Banks offers a wide range of products, such as credits cards, mortgages, and savings accounts, to its customers. However, they are currently trying to improve an even customer service with personalized product recommendations. A way for Santander Bank to understand the needs of their customers could be through building predictive machine learning models to identify individuals who will obtain the bank's products. This project utilized different classifier algorithms with customers' information and their bank product behavior to predict if they will acquire a credit card. Data cleaning and preprocessing was done on customers' information and their bank product behavior data before being used to train classifier models. A grid search cross validation was used to tune the hyperparameter in order to determine the best classifier and its best parameters. Classifiers used in the grid search included Logistic Regression, Multi-Layer Perceptron, Decision Tree, Random Forest, and XGBoost. After training using the grid search, we found the best classifier with the highest accuracy and its best parameters. We used the best model, Random Forest, to predict the target. The model created from this project was able to correctly classify if a Santander customer will acquire a credit card.

Keywords: bank products, machine learning, logistic regression, multi-layer perceptron, decision tree, random forest, XGBoost, grid search, hyperparameter tuning

Table of Contents

1. Introduction **4**

1.1. Motivation *4*

1.2. Problem *4*

2. Data **5**

2.1. Summary of Dataset *5*

3. Methods **5**

3.1. Exploratory Data Analysis *5*

3.2. Data Cleaning *6*

3.3. Data Preprocessing *7*

3.4. Hyperparameter Tuning and Building Models *8*

3.4.1. Logistic Regression 8

3.4.3. Decision Tree 9

3.4.4. Random Forest 9

3.4.5. XGBoost 9

3.5. Model Selection *9*

3.6. Predicting Target *10*

3.7. Evaluating Final Model *10*

4. Results **10**

5. Discussion **11**

6. Conclusion **12**

7. References **14**

8. Appendix **16**

1. Introduction

1.1. Motivation

Retail banks have to increasingly rely on marketing and product innovation to grow the total value of their average customer. Customers are now more active users and demand more personalized product recommendations based on purchase history or similar customers.

Machine Learning techniques, together with big data, can provide the means to better profile the customer. We believe that analyzing customer behavior allows us to not only help better profile the customer, but also recommend the appropriate product to offer in the future.

Santander, a Spanish banking group, incorporates the Eurozones's largest bank in Banco Santander, S.A, becoming one of the world's largest banks. Currently, only a small amount of Santander customers receives many recommendations from the bank, while the majority seldomly receives product recommendations. This results in an uneven customer experience.

Our goal is to help Santander Bank build a more effective product recommendation system, using Machine Learning techniques, to better meet customers' needs and ensure equal customer satisfaction. One product that Santander Bank offers is credit cards. Credit cards provide valuable consumer protection that is not given by cash and debit cards. By understanding their customers, Santander Bank may be able to provide better customer service through personalized product recommendations, such as credit cards.

1.2. Problem

Using customer information and product behavior data from Santander Bank, can we determine if a customer will acquire a credit card?

2. Data

2.1. Summary of Dataset

Through Kaggle, Santander Bank provides 1.5 years of monthly customer behavior data from January 28th, 2015 to May 28th, 2016. For our project we will only be using the train.csv file provided. This dataset contains 13,647,309 observations and 48 columns. Among the columns, 24 are the customers' information, and the other 24 are records of the products the customer has. The columns for the customer's information are: date, customer code, employee index, customer's country residence, customer's sex, age, date customer became first holder of a contract at bank, new customer index, customer seniority, primary customer code, last date as primary customer, customer type at beginning of month, customer relation type at beginning of month, residence index, foreigner index, if customer is spouse of an employee, channel used by the customer to join, deceased index, address type, province code, province name, activity index, gross income of the household, and segmentation. The possible products are: savings account, guarantees, current accounts, derivada accounts, payroll accounts, junior accounts, más particular account, particular account, particular plus account, short-term deposits, medium-term deposits, long-term deposits, e-account, funds, mortgage, pensions1, loans, taxes, credit cards, securities, home, account, payroll, pensions2, and direct debit.

Source: https://www.kaggle.com/c/santander-product-recommendation/overview

3. Methods

3.1. Exploratory Data Analysis

For exploratory data analysis, we checked for null values and NaNs, as well as noise in the data by looking for outliers using an outlier plot. Three features (customer seniority at the bank,

salary of customer, and age) were used in the outlier plots shown in figures 1, 3, and 5, respectively, in the Appendix. For all three plots, outliers were present, and this caused the distribution of the data to be widely spread out. We also looked at the categorical features and their number of unique values. To prepare for feature selection, we decided to look at the multicollinearity between the features by creating a correlation heat map shown in figure 9 in the Appendix. From this heatmap, we were not able to deduce the multicollinearity between features and this was because of the extremely high number of features in the plot. We created an 'eda.py' file that contains all the EDA classes and functions that will be called in our 'main.py' file.

3.2. Data Cleaning

For our project, we use the 'credit cards' column as our target and the rest of the columns as our features. Unable to work with the whole dataset due to the computational cost, we randomly sampled 250,000 observations with 'credit cards' = 0 and 250,000 observations with 'credit cards' = 1; where 0 implies customers without credit cards and 1 implies customers with credit cards. We appended the observations to create a data frame with 500,000 observations. We dropped columns, such as, the date, date customer became first holder of a contract at bank, last date as primary customer, customer code, address type, and province name. Next, we renamed the product columns to be more comprehendible, based on information given on the Kaggle website. We replaced null values with NaN and dropped the rows containing missing values. Additionally, we mapped the target values to be consistent with the rest of the values in their columns. For instance, we changed object values like '1.0' to 1 because we were unable to easily change the data type. Finally, we changed the data types of the features so they will be properly encoded in preprocessing. The data cleaning classes and functions are defined in a 'preprocessing.py' file and called in the 'main.py'.

3.3. Data Preprocessing

For preprocessing, we removed rows that we determined were outliers from our outlier plots. We re-plotted the outlier plots in figures 2, 4, and 6 in the Appendix to confirm this step. We one-hot encoded our categorical features with pd.get_dummies() and label encoded our target with LabelEncoder(). We did not run an imputation on the data since all rows with missing values were dropped. We did feature selection to reduce the dimensionality of our dataset with minimum loss of information. This allowed us to reduce the computational cost, train the models faster. We plotted the heat map again, shown in figure 10 in the Appendix, and from this heatmap, we were able to distinguish which features were highly correlated with one another. We were also able to see the feature importance, a list of the features sorted in the order of its importance, by running a RandomForestClassifier() on our dataset. From the feature importance shown in figure 7 in the Appendix, we were able to identify the features with the lowest scores. Those features with very low importance scores were dropped. We plotted feature importance again and can be seen in figure 8 in the Appendix.We did a train test split on our X and y to get X_train, X_test, y_train, and y_test. We checked the unique values for the target and saw an imbalance of the classes. To fix the imbalance of classes, we used RandomOverSampler() and fit our training data. We random sampled after splitting into training and testing because oversampling before splitting can cause the presence of the same observations in both train and test. We want our test data to remain unseen. The data preprocessing classes and functions are defined in the ‘preprocessing.py’ file and called in the ‘main.py’

3.4. Hyperparameter Tuning and Building Models

We used a combination of Pipeline and GridSearchCV to fine tune the hyperparameters of 5 classifiers. These classifiers are: Logistic Regression, Multi-layer Perceptron, Decision Tree, Random Forest, and XGBoost.

First, we created a dictionary of the classifiers with the key as the acronym of the classifier and the value as the classifier. Then, we created a dictionary of the pipeline with the key as the acronym of the classifier and the value as the pipeline with StandardScaler() and the classifier. Next, we created a dictionary of the parameter grids with the key as the acronym of the classifier and the value as the parameter grid of the classifier. In this parameter grids dictionary, each classifier has a parameter grid which contains the hyperparameters we want to fine tune.

For hyperparameter tuning we used GridSearchCV, which is an exhaustive search over the combinations of the specified parameter values for an estimator defined in the parameter grids dictionary. We also used StratifiedKFold which is a stratified K-folds cross validator. The K-fold cross validator divides the data into folds and ensures that each fold is used as a testing set at some point in the grid search. We set the K-fold n_splits = 10. Cross validation is useful for assessing the effectiveness of the models.

After running the grid search on our X_train and y_train, we created a list that appends the best score and parameters for each model/estimator. The hyperparameter tuning and model building classes and functions are defined in a ‘models.py’ file and called in the ‘main.py’

3.4.1. Logistic Regression

For fine tuning the parameters for logistic regression, we created a parameter grid with two multi-class options: ovr and multinomial. For multiclass: ovr, solver: [newton-sg, lbfgs, liblinear,

sag, saga], and C: [10 ** i for i in range(-4, 5)]. For multiclass: multinomial, solver: [newton-sg, lbfgs, sag, saga], C: [10 ** i for i in range(-4, 5)]. All the other parameters are set to the default.

3.4.2. Multilayer Perceptron

We created a parameter grid with hidden_layer_sizes: [10, 100] and activation: [identity, logistic, tanh, relu]. All the other parameters are set to the default.

3.4.3. Decision Tree

For the decision tree parameter grid, we set min_samples_split: [2, 10, 30] and min_samples_leaf: [1, 10, 30]. All the other parameters are set to the default.

3.4.4. Random Forest

For the random forest parameter grid, we set n_estimators: [10, 100, 1000], min_samples_split: [2, 10, 30], and min_samples_leaf: [1, 10, 30]. All the other parameters are set to the default.

3.4.5. XGBoost

In the XGBoost parameter grid, we set the parameters as eta: [10 ** i for i in range(-4, 1)], gamma: [0, 10, 100], and lambda: [10 ** i for i in range(-4, 5)]. We ran XGBoost a second time with K-fold n_splits = 5. All the other parameters are set to the default.

3.5. Model Selection

In model selection, we sorted the list of the best scores and parameters for each model. From the sorted list, we identified the model with the best score and the parameters associated with that score that will be used for the final model. The model selection class and functions are defined in the ‘models.py’ file and called in the ‘main.py’

3.6. Predicting Target

Using the best classifier: random forest with the best parameters: min_samples_leaf = 1, min_samples_split = 2, and n_estimators = 1000, we ran the model on our X_test to make a prediction.

3.7. Evaluating Final Model

We use a confusion matrix and ROC-AUC curve to evaluate the performance of our final model on our test dataset.

4. Results

The results from the grid search cross validation is presented below.

Models	Best Parameters	K-Fold CV Accuracy
Random Forest	min_samples_leaf: 1 min_samples_split: 2 n_estimators: 1000	91.7%
Decision Tree	min_samples_leaf: 1 min_samples_split: 2	89.4%
Multi-Layer Perceptron	activation: logistic hidden_layer_sizes: 100	87.2%
XGBoost	k-fold: n_splits = 5 eta: 0.0001 gamma: 10 lambda: 0.0001	86.3%

XGBoost	k-fold: n_splits = 10 eta: 0.0001 gamma: 0 lambda: 0.0001	86.3%
Logistic Regression	C: 0.0001 multi-class: ovr solver: liblinear	85.3%

Based on the results of our grid search, random forest is the best classifier using the parameters: min_samples_leaf = 1, min_samples_split = 2, and n_estimators = 1000, with an accuracy of 91.7%

After running our final model with the test data, we got a precision of 89.2%, recall of 95.4%, and F1 score of 0.922. From our test dataset, the true positives were 82,295, true negatives were 65,554, false negatives were 3,945, and false positives were 9,981. The final model's mean ROC (area =0.97).

5. Discussion

We selected random forest as the classifier for our final model since it had the highest accuracy from the cross-validation using the training dataset. Random forest is a good classifier (91.7%) since it works well with high dimensional data and is not as sensitive to overfitting compared to the other classifiers. Random forest also runs in parallel compared to XGBoost, which runs sequentially, making the computational time faster.

For the most part, the final model was able to accurately predict who got a credit card and who did not. However, it did make some wrong predictions which resulted in more false positives than false negatives. This means that it is predicting a larger number of customers that will get a credit card when in reality they did not. Furthermore, it is not predicting the customers that did get a credit card. The AUC score is high indicating that the model was able to correctly classify the class labels. Further research would be needed to conclude if Random Forest performs better than other classifiers.

Our project has a number of limitations. First, we only worked with a small subset of the whole data set. Perhaps if we used the whole dataset of over 13 million observations, we would be able to train our model better and achieve a higher accuracy. Our dataset may also still have noise and may need further cleaning and better reduction in dimensionality and selection of features. The grid search may also produce a different 'best model' and 'best parameters' if it used a much larger dataset. We also did not have enough time to run the grid search several times to fine tune the parameters for each classifier. There may be parameters that we have not tried that could have resulted in a better model.

In the future, we would like to be able to use the whole dataset of more than 13 million observations to create a model. We will also take the time to thoroughly clean our data, select our features, and tune our hyperparameters to accurately predict which customers will obtain a credit card. Eventually, we would like to be able to predict which other Santander Bank products the customers will get.

6. Conclusion

In conclusion, it is possible to use machine learning techniques to predict if a customer would get a credit card or not from Santander Bank. Further research would be necessary to apply

similar models to create a better personalized product recommendation system. Also, preprocessing is an important part of machine learning.

7. References

3.2.4.3.1. sklearn.ensemble.RandomForestClassifier¶. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

3.3. Metrics and scoring: quantifying the quality of predictions¶. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/model_evaluation.html.

Demonstration of multi-metric evaluation on cross_val_score and GridSearchCV¶. (n.d.). Retrieved from https://scikit-learn.org/stable/auto_examples/model_selection/plot_multi_metric_evaluation.html.

Hagan, M. T., Demuth, H. B., Beale, M. H., & Jesús Orlando De. (2016). *Neural Network Design*. S. l.: s. n.

Mani, K. (2019, February 17). Introduction to Exploratory Data Analysis. Retrieved from https://medium.com/datadriveninvestor/introduction-to-exploratory-data-analysis-682eb64063ff.

Narkhede, S. (2019, May 26). Understanding AUC - ROC Curve. Retrieved from https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

Peng, R. D., & Matsui, E. (2016). *The Art of Data Science*. Skybrude Consulting, LLC.

Raschka, S., & Mirjalili, V. (2017). *Python Machine Learning*(2nd ed.). Packt Publishing.

Ravanshad, A. (2019, August 1). Gradient Boosting vs Random Forest. Retrieved from https://medium.com/@aravanshad/gradient-boosting-versus-random-forest-cfa3fa8f0d80.

Santander Product Recommendation. (n.d.). Retrieved from https://www.kaggle.com/c/santander-product-recommendation/data.

sklearn.linear_model.LogisticRegression¶. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

sklearn.neural_network.MLPClassifier¶. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.

sklearn.tree.DecisionTreeClassifier¶. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.

VanderPlas, J. (n.d.). Hyperparameters and Model Validation. Retrieved from https://jakevdp.github.io/PythonDataScienceHandbook/05.03-hyperparameters-and-model-validation.html.

XGBoost Parameters¶. (n.d.). Retrieved from https://xgboost.readthedocs.io/en/latest/parameter.html.

8. Appendix

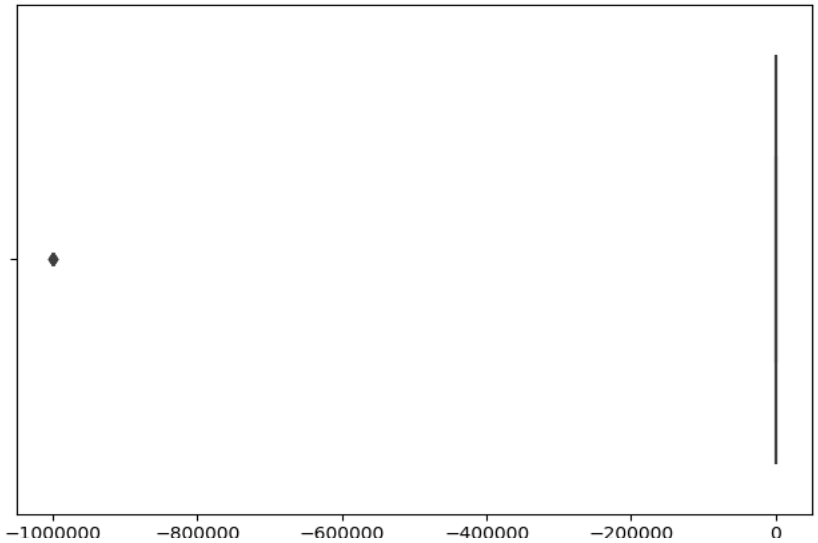

Figure 1. Outlier plot of customer seniority before feature selection

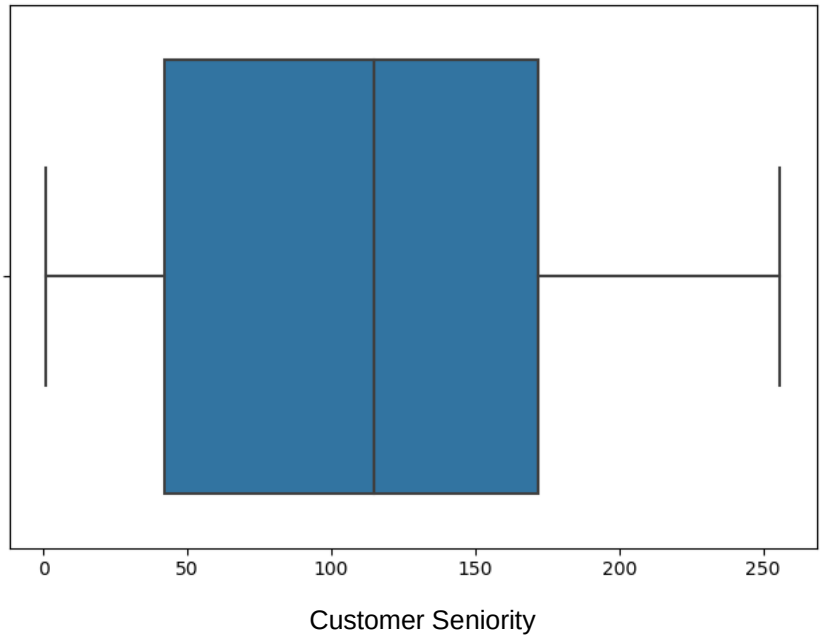

Figure 2. Outlier plot of customer seniority after removing outlier.

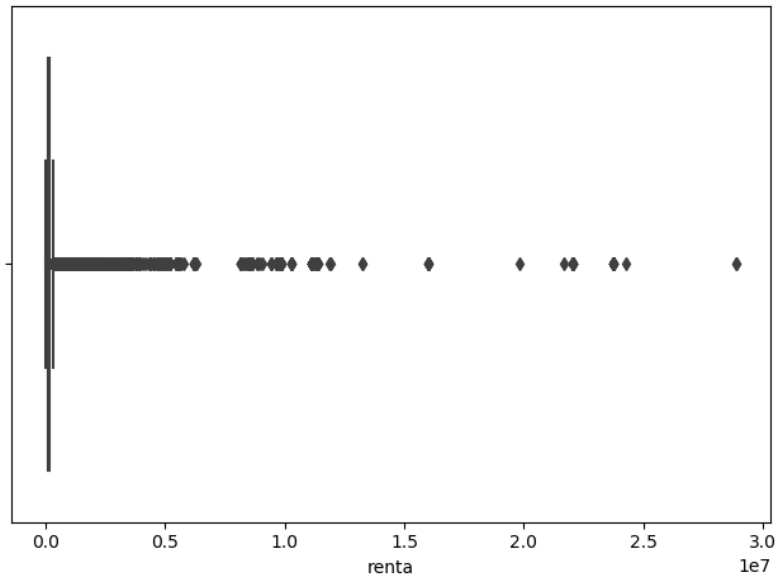

Figure 3. Outlier plot of customer renta (salary) before removing outliers.

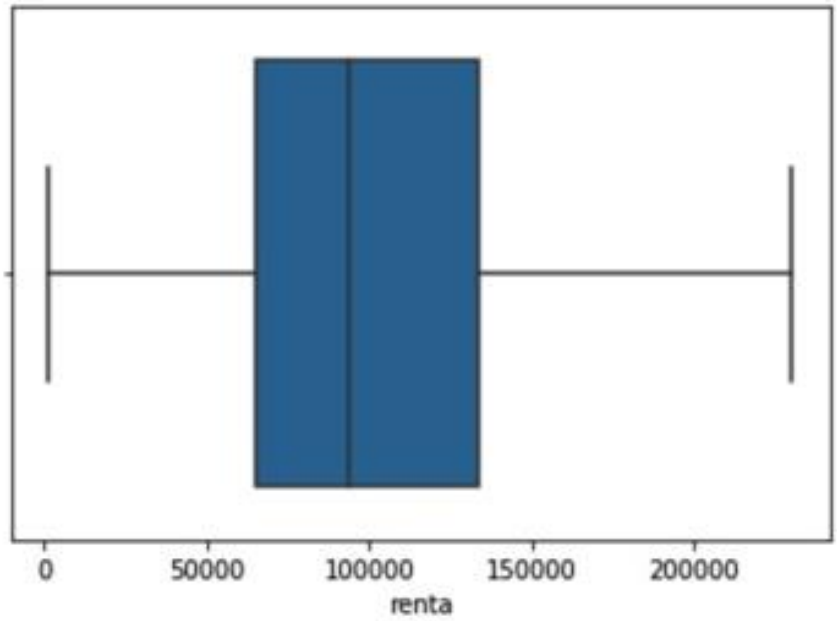

Figure 4. Outlier plot of customer renta (salary) after removing outliers.

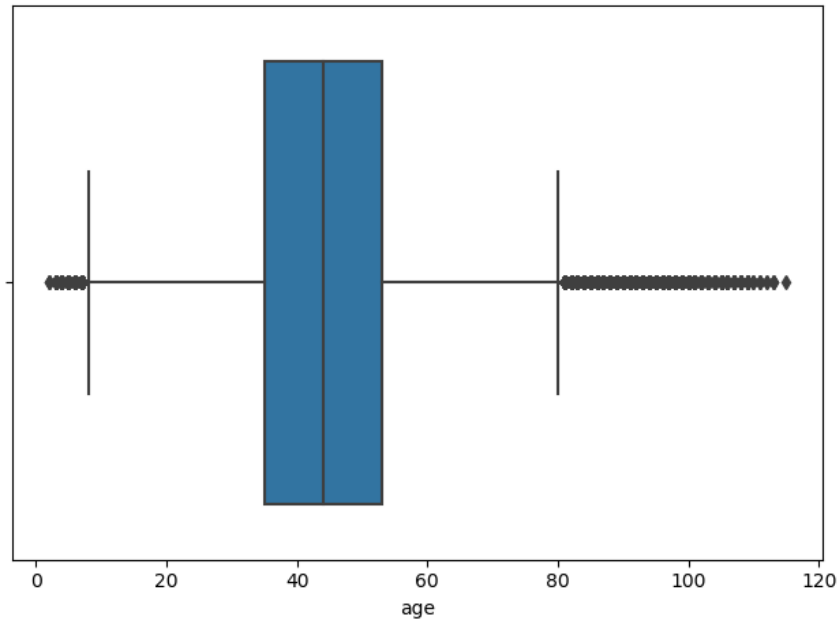

Figure 5. Outlier plot of customer age before removing outliers.

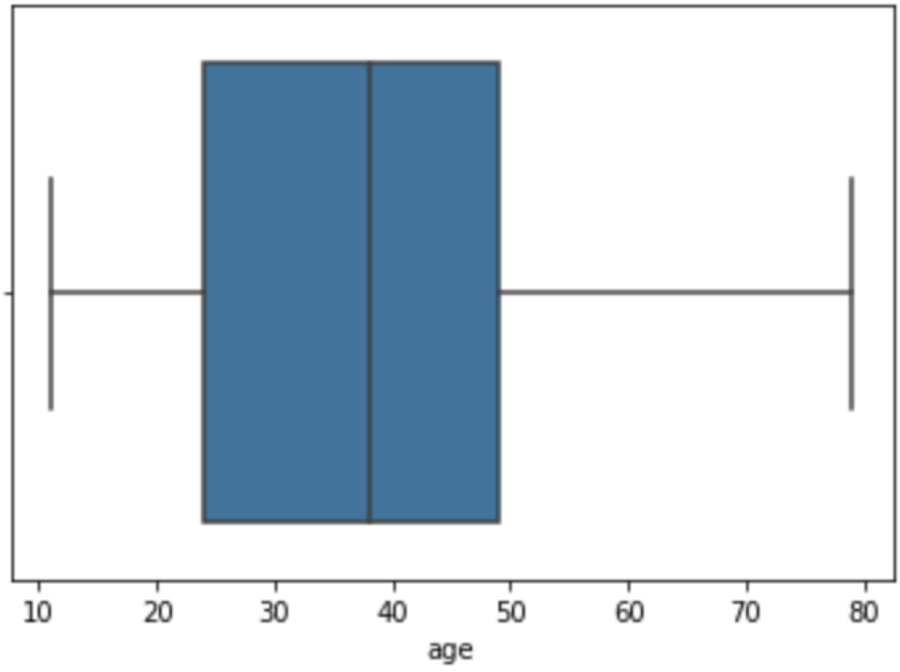

Figure 6. Outlier plot of customer age after removing outliers.

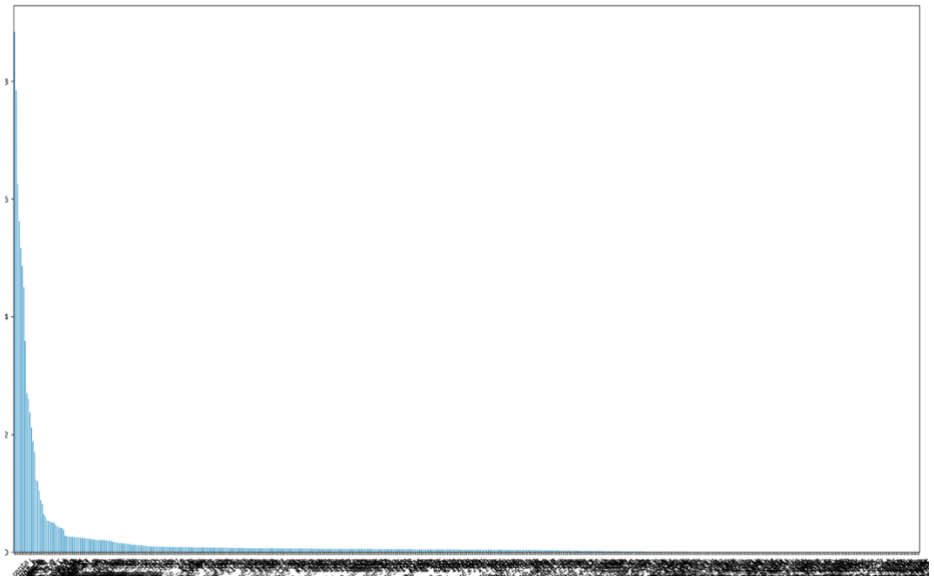

Figure 7. Feature Importance before Feature Selection.

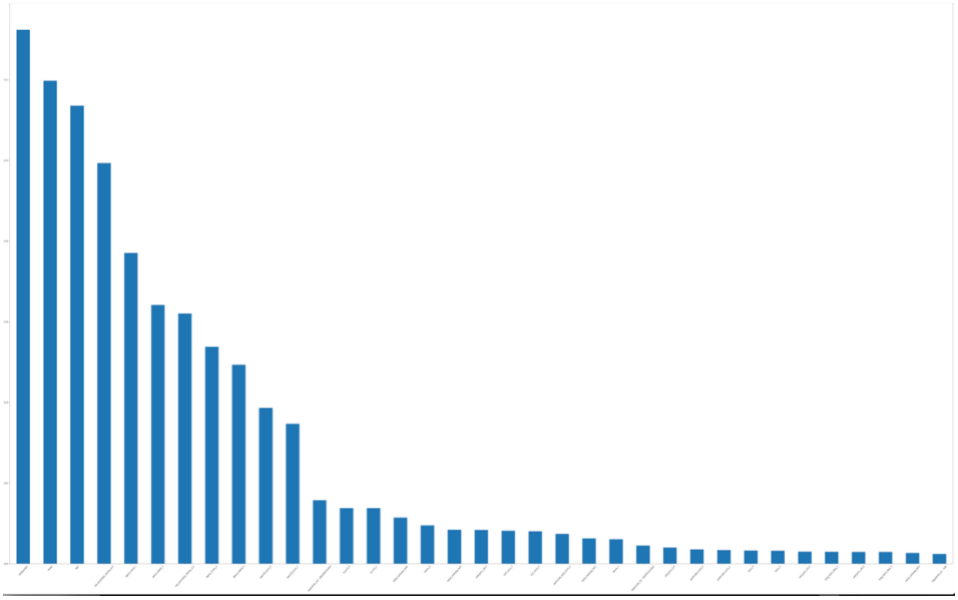

Figure 8. Feature Importance after Feature Selection.

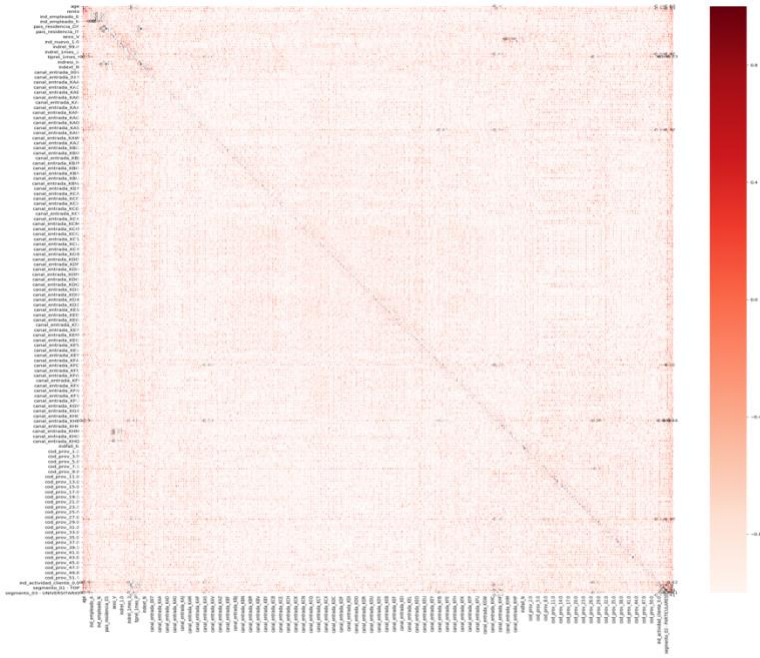

Figure 9. Heat Map before Feature Selection

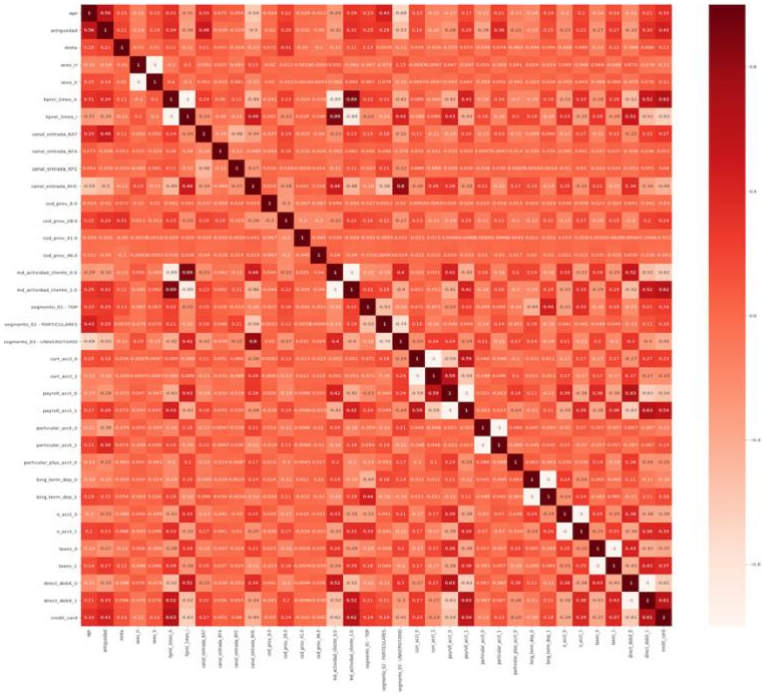

Figure 10. Heat Map after Feature Selection.

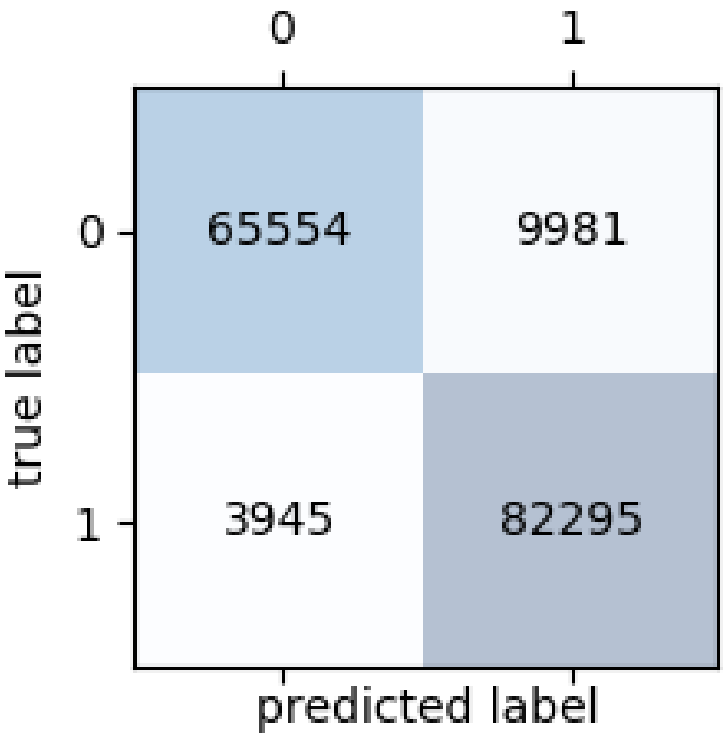

Figure 11. Confusion Matrix of Predicted Label and True Label

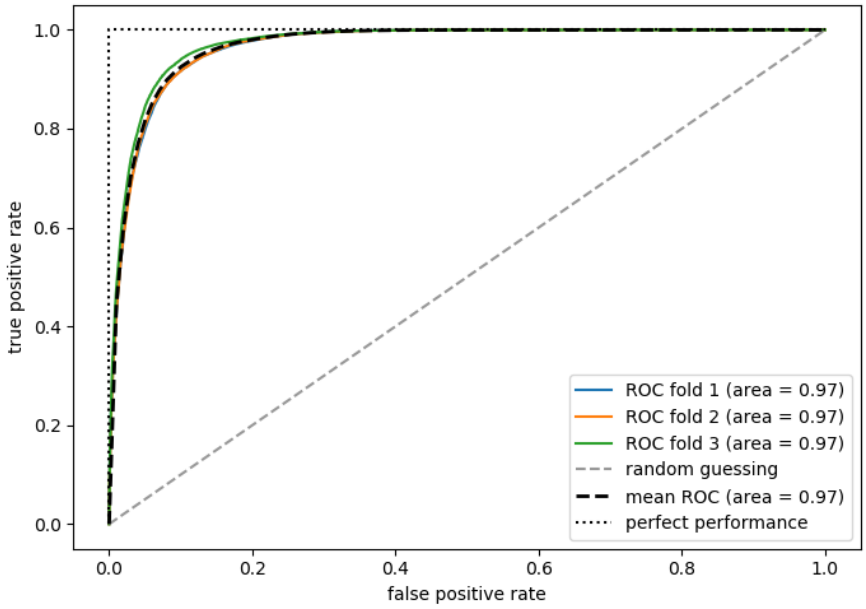

Figure 12. ROC-AUC Curve