Response Model: A Real-World Binary Classification Problem

Data Science and Advanced Analytics

Machine Learning

Rita Franco	m20180080
Rodrigo Umbelino	m20180060
Vitor Manita	m20180054

Abstract

The aim of this project is to build a predictive model that will increase the profit of the marketing campaign of a fictional company. In order to achieve this, we applied several techniques of data preprocessing, feature engineering and Machine Learning models that we have learned during the semester. The result should be a model that will allow the company to perform targeted marketing to customers that are likely to accept the campaign, leaving out the remaining non-respondents.

1. Sample

Given the nature of the project, the dataset that we worked on was provided to us. In order to better grasp all the techniques that we would have to implement, we defined three clear project phases. The first one, the exploration phase, is the phase where we understood the shape and scope of the dataset. Following this step, we proceeded to the feature engineering phase. In this phase, we defined new variables and transformed already existent ones, with the objective of producing even more significant data.

Lastly, our final phase of model analysis and implementation. This phase consisted mainly in balancing our dataset and explore several models. In addition to this, we also implemented different methods in order to improve our results.

With the initial provided dataset, our first step was to split it into training and test data. With a total of 2240 observations, we decided that 80% of those (1792 observations) would constitute the training set, and the remaining 20% (448 observations) would integrate the test set. The following preprocessing and feature engineering phases were only applied to our training set, leaving the test set intact for model testing.

2. Explore

Our first approach in the exploration phase was to evaluate the sixth marketing campaign response rate in the training set, which resulted in a value of 15%. After a brief exploration, the presence of null values caught our attention, since it represents a concern in the context of the problem, therefore we analyzed the number of missing values per column and its overall percentage when comparing to the entire dataset. We concluded that the variable Income was the only one that had missing values. In our first run, 20 observations were classified as missing values, representing around 1.12% of the training dataset. Taking into consideration the "good measure" threshold of a 3% cutoff line on missing values, we decided to remove these observations from the dataset, since there might be a case of not so trustworthy data regarding these specific observations.

Besides the missing values, we also checked the variance for each numeric column in the dataset. For columns with variance close to 0, namely **Z_CostContact** and **Z_Revenue**, we considered them as metadata, removing them from our dataset since they do not contribute with any discrimination ability towards the target variable.

In addition to the variance check, we also thought that there could possibly be erroneous negative numeric values in the dataset, which would not make sense in the context of any of our variables. After checking, we concluded that there were none.

The next logical step would be to analyze the categorical variables. We were able to quickly identify attributes that did not make sense for the context of the problem, particularly in the **Marital_Status** variable (*Absurd*, *Alone* and *YOLO*). Since these observations were only 6 observations, we removed them, following the same logic of the missing values in the numeric variables, given that there also might be a case of not so trustworthy data.

Another point that we had to consider was the treatment of duplicate variables. After some thought, we decided to only check for duplicate observations that had different values between them in the **Response** column (20 observations), and consequently remove them, since they can contribute with opposite information to our model. On the other hand, we chose not to delete observations that are duplicates even in the **Response** column, since classifiers act on probability and expectation. If we did remove them, we could risk reducing the weights that a certain customer's characteristics and behaviors have on the target. It is important to notice that this problem has no right solution, since its resolution would require a better business insight, and duplicates can occur again in unseen data. In the end, we believe our model would generalize better by taking this decision.

Lastly, we studied the correlation amongst the independent variables and the dependent one. In the available data, we can observe that there are some strong associations between features, allowing us to think in future decisions when approaching variable selection, since highly correlated variables carry with them the problem of model multicollinearity, where some variables will appear to have a higher predictive power to explain the target variable. However, this can be just an illusion caused by the redundancy of information variability.

3. Modify

Beginning our feature engineering phase, our training set now has 1746 observations, due to the preprocessing steps that were applied previously. In this phase, the goal was prepare the dataset as best as possible to provide the best input to the models.

3.1 Outliers

The first procedure was outlier detection and removal. We began by plotting the distribution of each variable in the dataset, as well as the **Response** column value within the scope of the variable's distribution. In hindsight, we could easily detect some outliers just by looking at the variables' distribution.

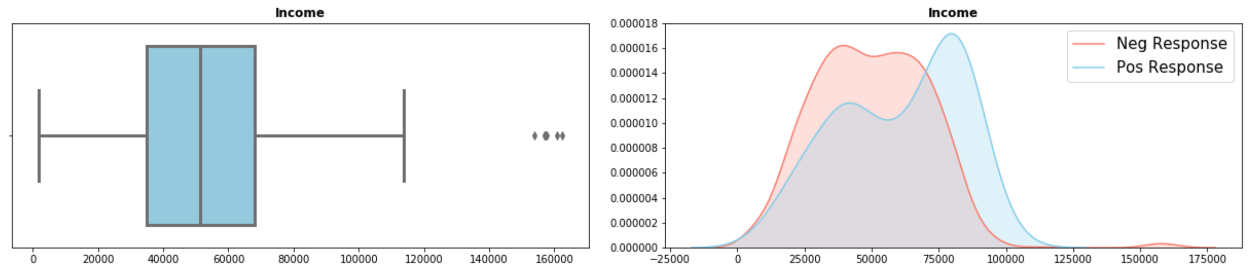

Figure 1. Income distribution boxplot (left) and density kde plot (right)

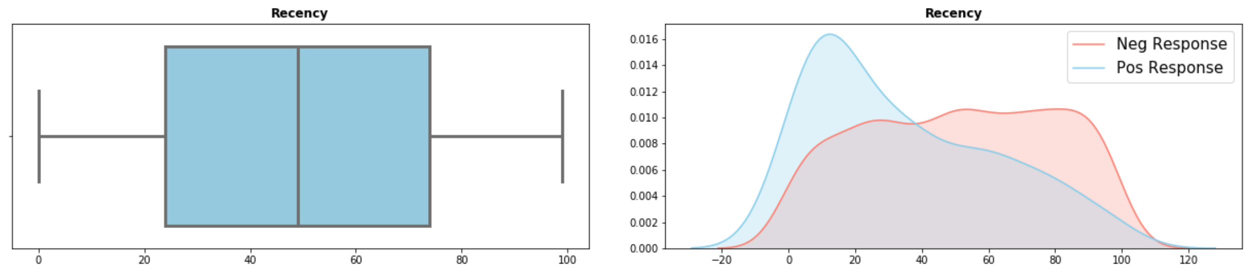

Figure 2. . Recency distribution boxplot (left) and density kde plot (right)

Our strategy was to implement multiple outlier detection methods and then rank them from best to worst according to different scoring methods (recall, accuracy, precision and f1-score) using a Logistic Regression to predict on validation data. All the implemented outlier detection methods are listed in the table1.

Methodology	Description
Interquartile Range	Characterized by the difference between the 1st and the 3rd quartiles, which can summarize variability in a dataset with outliers, on feature level
Standard Deviation	Characterized by the calculation of the mean and standard deviation, identifying data points as outliers which are at a certain threshold number of STDs away, on feature level
DBSCAN	Density-based clustering algorithm that aggregates points that are close together, and marks data points that are in low-density areas as noise/outliers
Isolation Forest	Algorithm that isolates observations by randomly selecting a feature and using recursive partitioning and the path length as the decision function and measure of normality
Local Outlier Factor	Algorithm that compares the density of a point to the densities of its neighbours, data points that have significantly lower values than its neighbours are considered outliers
Mahalanobis Distance	Characterized by the calculation of the mahalanobis distance between all points in all features and identifying data points as outliers which are at a certain threshold number of standard deviation away

Table 1. Summary of Outliers Removal methods

It is important to notice that the Interquartile Range and Standard Deviation methods were applied in synchronicity, meaning that we only took into consideration outliers for each column that were identified by both methods simultaneously. All the implemented methods and respective scores are listed in the table 2.

Methodology	Recall	Accuracy	Precision	F1-Score
Mahalanobis Distance with 4 Standard Deviations	0.3638	0.8881	0.7562	0.4905
Mahalanobis Distance with 3 Standard Deviations	0.3663	0.8895	0.7439	0.4896
Mahalanobis Distance stacked with 4 Standard Deviations	0.3635	0.8889	0.7474	0.4878
DBSCAN	0.3554	0.8840	0.7121	0.4728
Isolation Forest	0.3364	0.8975	0.7748	0.4625
3 Standard Deviations intersected with 1.7 Interquartile Range	0.3051	0.8979	0.7578	0.4307
Local Outlier Factor	0.2811	0.8889	0.7189	0.4035

Table 2. summary of Outliers Removal methods metrics

Considering that the Mahalanobis with 4 Standard Deviations was the method that provided the best results, we used it to remove the outliers. A total of 13 outliers were removed by this methodology, resulting in a decrease to 1733 observations in total.

3.2 Imputing missing values

Having dealt with the outliers, we implemented a couple of methods to imputate the missing values in the test set. Regarding the numeric variables, two approaches were used: a basic imputation method and a model based imputation method. The basic imputation method consisted in replacing the null values of the dataset by a strategy of our choice (mean, median or mode). The model based imputation method consisted in training several models to predict the missing values per column, calculating the mean squared error and returning the best imputation model per column. This method was superior in performance when comparing to the basic imputation method, therefore we applied it to the test set. The result was the replacement of the missing values in the **Income** column using a DecisionTreeRegressor, which was the best model for that column returned by the methodology.

Regarding the categorical variables, we implemented a method that would imputate possible missing values with the mode of the null value column.

3.3 Variable Creation and Transformation

The next step in the feature engineering phase was to analyze the existent variables, keeping in mind the context of our problem, and transform them and create new variables that we deemed that it would bring explanatory power to our model. All the variables that were transformed and the ones that were created and introduced in the dataset are listed in the table 3.

Variable	Description	Min	Max	Mean
Age	Age of the customer	18.00	121.00	45.37
Days_as_cust	Number of days that a given individual has been a company's customer	0.00	699.00	353.50
Mnt_tot	Sum of all monetary purchases per customer	5.00	2525.00	609.66
Frq	Total number of purchases made by a customer	0.00	32.00	12.55
Childnum	Total number of children that a specific customer has	0.00	3.00	0.95
R_MntFrq	Monetary Frequency Ratio – given by the sum of monetary purchases divided by the purchase frequency	2.67	187.67	37.55
R_MntIncome	Monetary Income Ratio – given by the sum of monetary purchases divided by the customer's income	0.00	0.04	0.01

AcceptedTot	Total number of accepted campaigns per customer	0.00	4.00	0.30
R_DealFrq	Discount Purchases Ratio – given by the number of discount purchases divided by the purchase frequency	0.00	0.79	0.24
RFM	Recency, Frequency and Monetary Value – a score that represents customer value	111.00	555.00	335.95
R_Mnt[Product]	Mnt[Product] divided by Mn_tot			
R_Mnt_[Channel]	Mnt[Chanel] divided by Mn_tot			
R_Num_[Chanel]Purchases	Num[Channel]Purhases divided by Frequency			

Table 3. New and transformed variables

3.4 Coherence Checking

After the creation and transformation of variables, we decided that it would be helpful to perform some incoherence checking. To achieve this, we created a few coherence rules so that we could see if all the data made sense in the problem's context:

1) If the sum of all monetary purchases of a customer (**Mnt_tot**) is greater than 0, the total number of purchases of that customer (**Frq**) cannot be equal to 0.
2) The year of birth (**Year_Birth**) of a customer cannot be after the date that that customer first became a customer (**Dt_Customer**).
3) The total number of purchases with a discount (**NumDealsPurchases**) made by a customer cannot be smaller than the total number of purchases of that customer (**Frq**).
4) If the number of days since the last purchase of a customer is greater or equal to 0 (**Recency**), the sum of all monetary purchases of that customer (**Mnt_tot**) cannot be equal to 0.

Then, there was only 1 record that did not meet the coherence criteria. Therefore we removed it from the dataset.

3.5 Categorical Transformation

Having dealt with the numerical variables, it was time to analyze the categorical variables. Firstly, we plotted the proportion of respondents (**Response** column) for each attribute in the categorical variables (**Education** and **Marital_Status**), defining a threshold that represents the overall acceptance rate in the dataset (15%). Regarding **Education**, customers that have a *PhD* or *Master's* degree revealed to have a higher acceptance rate than all other education levels. On a civil status perspective, customers who are *Single*, *Widow* or *Divorced* also have a higher acceptance rate than the decided threshold, while those who are *Together* or *Married* appear to be below it. After this analysis, we proceeded to encode these variables according to their position regarding the threshold: **Education** and **Marital_Status** are now represented by 1 for categories whose acceptance rate is, individually, superior to 15%, and by 0 for categories whose acceptance rate is inferior to 15%.

The last step in our categorical variables encoding was to make sure that we removed possible strange categories from the test set that would not make sense, as there were some initially in the training set, by automatically creating a dictionary with the right replacements. However, we paid attention to strange categories in unseen data that, since they cannot be deleted, they are replaced by the value 0.

3.6 Feature Selection

Another important point to consider in any Machine Learning project is feature selection. After feature extraction, we have around 90 features and, as pointed out in the previous chapter, the correlation between some of them will decrease the quality of our model. By only selecting the most discriminant variables, we can avoid the curse of dimensionality and simplify the analysis. With this goal in mind, our strategy was to run several methods to access feature importance and rank them in a *DataFrame*, where we can later vote the most important ones and evaluate their validity. All methods used to compute feature importance are listed in the table 4.

Methodology	Description
Chi-Squared Test	Statistical test of independence to infer the dependence between two variables.
Linear Regression	Use Linear Regression's R-squared measure for feature ranking.
Random Forest	Use Random Forest's feature importance measure for feature ranking.
RFE (Logistic Regression)	Recursive Feature Elimination using features' weights provided by the Logistic Regression model.
Extra Trees	Use Extra Trees' feature importance measure for feature ranking.
Decision Tree	Use Decision Tree's feature importance measure for feature ranking.
WOE and IV	Use Weight Of Evidence and Information Value to explain the predictive power of variable and rank it.
Eli5 Permutation	Algorithm that computes feature importance by measuring the estimator's decrease in score when that variable is non-existent .

Table 4. summary of Feature selection technniques

After ranking all feature importance methods, we used their correlation to decide which variables to use. If the most important feature, according to the rank, was highly correlated, this is, if a correlation higher than 70% existed, with another feature, we would drop the least important one. This feature selection technique produced a result of 27 features from the original dataset.

Another way to try to reduce the dimensionality of our data was by transforming all features into their Weight of Evidence (woe) and extract the Variance Inflation Factor for each one. Using our earlier feature ranking technique, we extracted the top 15 variables to proceed with this analysis and, using the good practice rule of thumb of a 10 VIF score, we iteratively removed features until the remaining ones had a smaller score than this threshold, ending up with 10 final features.

Our third and last feature selection approach was implementing Recursive Feature Elimination with Cross Validation (RFECV). This method fits a model, in our case we used a simple Logistic Regression, and removes the weakest features recursively until the optimal number of features is reached, according to the score of the algorithm. This method has its advantage of automatically selecting the number of variables, however, we lose some control over this extraction.

After all these techniques were implemented and tested, we compared them according to multiple scoring metrics. The final results are presented in the table below:

Given the results in table above, we opted to use the removal of correlated features automatically by using the technique that performed the better within 10 folds of stratified cross-validation.

Features	VIF
AcceptedTot	4.2843
Days_as_cust	4.0770
R_Mnt_NumStorePurchases	2.7437
Recency	3.5019
AcceptedCmp5	2.6517
MntMeatProducts	8.7338
R_DealFrq	4.5130
R_MntMeatProducts	8.0887
R_NumCatalogPurchases	4.7085
R_MntFrq	9.7717
AcceptedCmp1	2.1562

Table 5. VIF scores (under 10) per variable

3.7 Feature Decomposition

Methodology	Description
PCA	Linear transformation technique that aims to reduce dimensionality by detecting correlation between variables.
Factor Analysis	Similar technique to PCA but with assumptions regarding the importance of features.
FastICA	Fast algorithm for Independent Component Analysis that reveals hidden factors within sets of random features.

t-SNE	Algorithm that computes affinities between datapoints and tries to keep them in a new space with reduced dimensions.
LDA	Similar technique to PCA that tries to maximize the separability between important features.
Kernel PCA	Variant of PCA that uses the Kernel trick to find the optimal representative direction in data, even if the principal components are non-linear.
Feature Agglomeration	Algorithm that agglomerates features by recursively clustering them.
Gaussian Random Projection	Method that projects a random lower-dimensional space using Gaussian Distribution.

Table 6. Summary of Feature Decomposition Techniques

After our feature selection analysis, it was time to test different feature decomposition techniques. Unlike feature selection, the goal of feature decomposition is to decompose the original set of features into several subsets, build a classification model for each of them, and combine all generated models [1]. Having this goal in mind, we applied several feature decomposition techniques to the dataset. We started with Principal Component Analysis, which resulted in 10 principal components with 80% of cumulative explained variance. The choice of using 10 components was also supported by analysis the explain variance of each. In turn, we used this result as an argument for the remaining feature decomposition methodologies, which are listed in the table 6.

After the implementation of all of these different feature decomposition methods, we followed the same logic as in the feature selection phase, and compared all of the methods according to different scoring metrics. We also thought that it would be interesting to compare our chosen feature selection method, that is what why the table 7 contains a record with our *variable_selection* in direct comparison with the feature decomposition techniques.

Methodology	Recall	Accuracy	Precision	F1-Score
WOE	0.5235	0.9036	0.7556	0.6157
Kernel PCA	0.4848	0.9019	0.7675	0.5924
PCA	0.4848	0.9019	0.7675	0.5924
Factor Analysis	0.3916	0.8892	0.7342	0.5078
Variable Selection	0.3462	0.8706	0.6102	0.4373
Feature Agglomeration	0.2867	0.8828	0.7933	0.4202
Gaussian Random Projection	0.2170	0.8666	0.6651	0.3257
FastICA	0.0000	0.8510	0.0000	0.0000
t-SNE	0.0000	0.8510	0.0000	0.0000

Table 7. Feature Decomposition evaluating metrics

As we can observe in the result table, there were in fact several feature decomposition methods that performed better than our top feature selection method. After giving some thought regarding which method to choose from, we decided to go with the RFECV. The reasons behind this decision were mainly because opting for a feature decomposition method would mean that we would lose a lot of interpretability power regarding the dataset's features, and would make a lot harder to grasp the results of our model.

4. Models

After having the prepared data ready to model, we began by exploring several models and how they they scored across accuracy, recall, precision and f1 - score. A big decision we add to make was to balance or not the training, since this one was highly imbalanced. To do this we resorted to SMOTE package that creates artificial samples of the data it receives using the K Nearest Neighbourhood algorithm. It is important to notice that, to evaluate and tune the parameters of the models, we use cross validation that, in each fold, splits a part of the training set as validation. So, if we generated artificial samples on the entire training set, we would be biasing the data that would later on be selected as validation, that would cause the overfitting of the model. To counter this effect, we had to build a pipeline that receives each model and only oversamples the fold selected as training, leaving the validation intact.

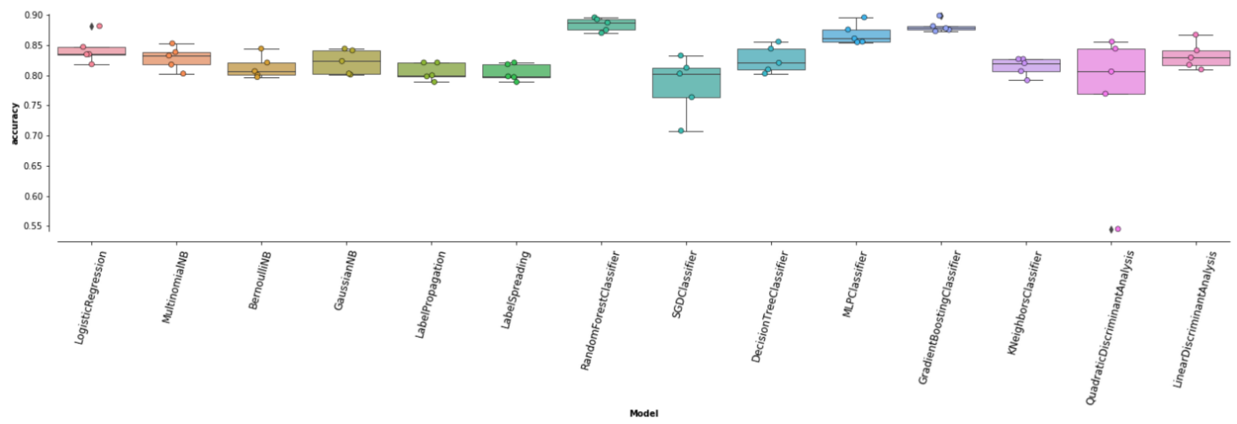

Figure 3. Boxplot of the distribution of overall Accuracy over 5 cross validation folds for each model

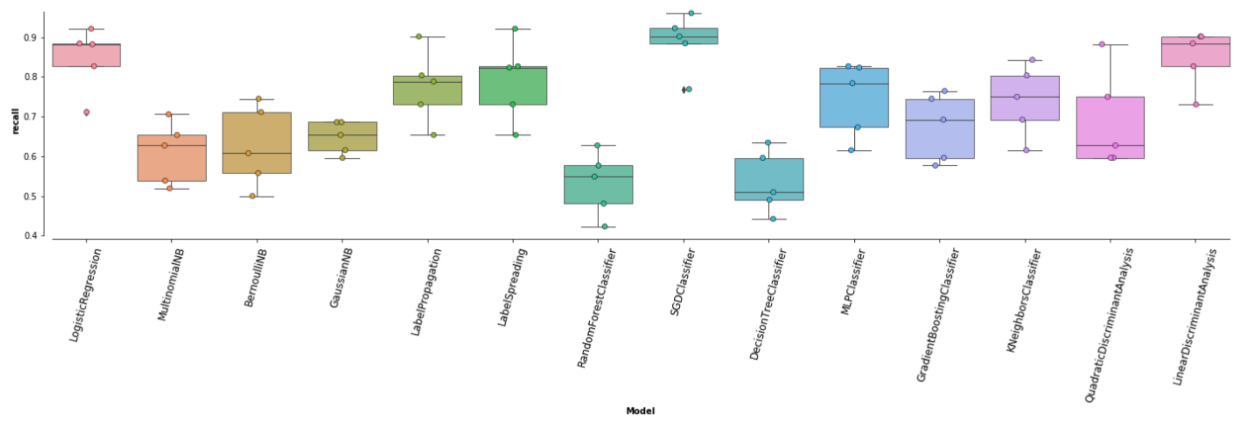

Figure 4. Boxplot of the distribution of overall recall over 5 cross validation folds, for each model

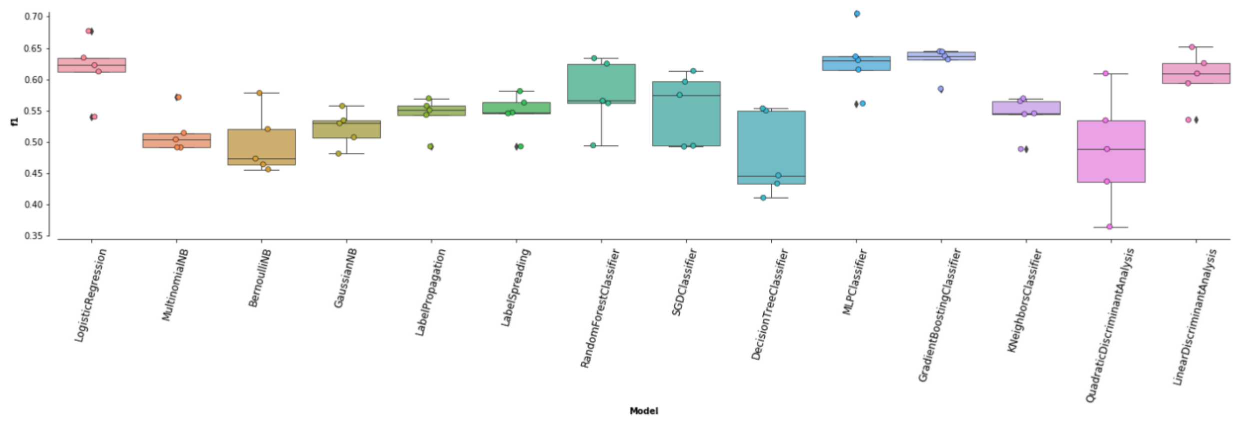

Figure 5. Boxplot of the distribution of f-1 score over 5 cross validation folds, for each model

To access all classification metrics, we experimented 20 different classifiers, but later focused on only some of them later on. By running a simple and basic classification of all our models, we can see that some of them, such as *RandomForestClassifier*, Logistic Regression and MLP are good candidates to proceed to fine tuning.

What we did was a cross validation that averages the scores of the folds and chooses K best models to fine tune. To achieve this, we used a grid search with several parameter combinations that we tried along the project.

After having the best combination of parameters for each one of the top selected classifiers, we decided to experiment some ensemble methods, namely Bagging and Voting. After trial and error we did not waste too much time on bagging since it was time consuming and did not promise better results. Voting, however, proved to use the best of each classifier and resulted in a better and final approach of our top models.

Implementing this voting ensemble on our final baseline, the final step was trying to maximize the profit, using the validation set and cross validations to achieve an average of the best activation threshold whereas the prediction of a customer would be one, being this 0.5 by default.

In the end of our runs, we concluded the following metrics:

- ACCURACY: 0.886 <<<
- TEST FINAL PROFIT: 302 €
- TEST FINAL NORMALIZED PROFIT: 56.34%
- AUROC SCORE: 0.9222587848160771
- PR SCORE: 0.6799934627793318

Then, we changed the binarization of the probability of acceptance of our predictions according to the average threshold activation and we predict that we would have a marketing campaign acceptance of 1426 clients, resulting in a profit of 11408 €.

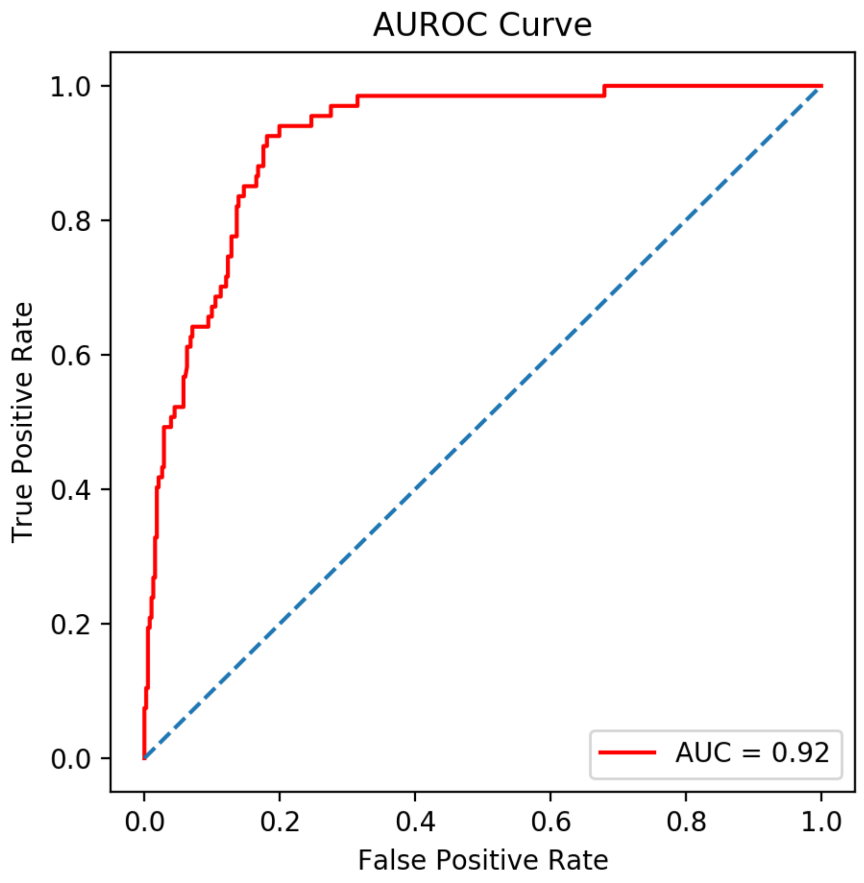

Figure 6. AUROC on final model

Label	precision	recall	f1-score	support
0	0.95	0.91	0.93	381
1	0.60	0.75	0.66	67
Micro AVG	0.89	0.89	0.89	448
Macro AVG	0.77	0.83	0.80	448
Weighted AVG	0.90	0.89	0.89	448

Figure 7. Metrics from final model on Test Data

5. Access and conclusions

This project was an excellent way of applying the Machine Learning course concepts that we learned throughout the semester. Our main difficulty was to understand how to synchronize the two environments that we had to work with, the Jupyter Notebook and the PyCharm pipeline. Although our work ran smoothly while working in Jupyter, we faced some adversities while transferring our work to PyCharm, since we had an hard time adapting our code to the pipeline. Our second difficulty was the time factor, maybe with a bit more time and practice we would be able to implement and test all the concepts that we planned or started to implement. However, we can confidently say that it was a very hands-on and efficient way of developing our Machine Learning knowledge and Python programming skills.On future objectives, we would like to explore more possibilities, namely evaluating if decomposition feature reduction would add more information to our models and we would like to apply the knowledge obtained in Computational Intelligence for Optimization to introduce genetic programming to our framework in order to have a more efficient and optimal hyperparameter search of the classifier models.

6. References

[1] O. Maimon and L. Rockach, "Improving Supervised Learning by Feature Decomposition," pp. pp. 178-196, 2002.

[2] s.-l. developers, "Plot Iterative Imputer Variants Comparison," 2007 - 2019. [Online]. Available: https://scikit-learn.org/dev/auto_examples/impute/plot_iterative_imputer_variants_comparison.html. [Accessed april 2019].

[3] s.-l. developers, "Unsupervised dimensionality reduction," 2007-2018. [Online]. Available: https://scikit-learn.org/stable/modules/unsupervised_reduction.html#feature-agglomeration. [Accessed april 2019].

[4] F. Malik, "Processing Data To Improve Machine Learning Models Accuracy," 29 november 2018. [Online]. Available: https://medium.com/fintechexplained/processing-data-to-improve-machine-learning-models-accuracy-de17c655dc8e. [Accessed april 2019].

[5] W. Koehrsen, "A Feature Selection Tool for Machine Learning in Python," 22 june 2018. [Online]. Available: https://towardsdatascience.com/a-feature-selection-tool-for-machine-learning-in-python-b64dd23710f0. [Accessed april 2019].

[6] S. Asaithambi, "Why, How and When to apply Feature Selection," 31 january 2018. [Online]. Available: https://towardsdatascience.com/why-how-and-when-to-apply-feature-selection-e9c69adfabf2. [Accessed april 2018].

[7] J. Brownlee, "8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset," 19 august 2015. [Online]. Available: https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/. [Accessed april 2019].

[8] S. Krishnan, "End to End—Predictive model using Python framework," 21 june 2018. [Online]. Available: https://towardsdatascience.com/end-to-end-python-framework-for-predictive-modeling-b8052bb96a78. [Accessed april 2019].

[9] S. Krishnan, "Weight of evidence and Information Value using Python," 29 april 2018. [Online]. Available: https://medium.com/@sundarstyles89/weight-of-evidence-and-information-value-using-python-6f05072e83eb. [Accessed april 2019].

[10] H. Le, "NeuroEvolution, NEAT Algorithm and My NEAT," 27 jully 2018. [Online]. Available: https://medium.com/datadriveninvestor/neuroevolution-neat-algorithm-and-my-neat-b83c5174d8b0. [Accessed april 2019].

[11] A. Gad, "Artificial Neural Networks Optimization using Genetic Algorithm with Python," 7 march 2019. [Online]. Available: https://towardsdatascience.com/artificial-neural-networks-optimization-using-genetic-algorithm-with-python-1fe8ed17733e. [Accessed april 2019].

[12] K. Arvai, "Fine tuning a classifier in scikit-learn," 24 january 2018. [Online]. Available: https://towardsdatascience.com/fine-tuning-a-classifier-in-scikit-learn-66e048c21e65. [Accessed april 2019].

[13] B. Shetty, "Supervised Machine Learning: Classification," 12 december 2018. [Online]. Available: https://towardsdatascience.com/supervised-machine-learning-classification-5e685fe18a6d. [Accessed april 2019].