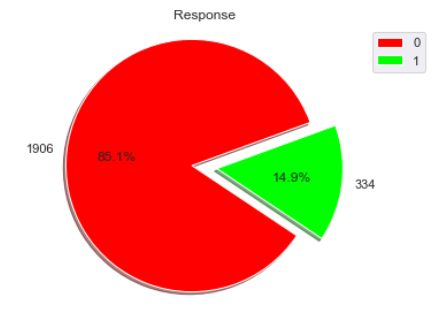
**INTRODUCTION**

This report is divided into two parts, the first one related with the data exploration and research in which we explain the things we had in consideration while doing our project. Also, we showed the initial research we made, and we give a brief explanation of what we implemented and why. The second part consists on our pipeline and the models and algorithms and combinations we used, we show some of the versions we decided to run as well as the benchmark and some conclusions we have reached. Our main objective is to maximize the profit of the next campaign the store will do.

**DATA EXPLORATION AND RESEARCH**

The first thing we did was data exploration, using the Jupyter Notebook, that helped us identifying missing values, seeing the type of variables that we were handling, understanding how imbalanced our dataset was and gathering some other insights about our data and model.

Our dataset consists of 2240 rows and 28 columns. Each row represents one client and each column an aspect of his behaviour from his spending habits to personal information. There are also two columns called Z\_CostContact and Z\_Revenue which have the same value for all clients, so we will drop them and keep only the values to do the profit calculations later on.

Based on the initial exploration, being that our response rate is 14.91%, we could understand that we have an **imbalanced dataset**. Which means that from our whole dataset we have 1906 clients with Response 0 and 334 clients with Response equal to 1. This can be a problem when training our model, because for us getting a 0 right is not the same as getting a 1 right, this last one is way more important. As we can see in [1], [2] there are several ways to deal with this:

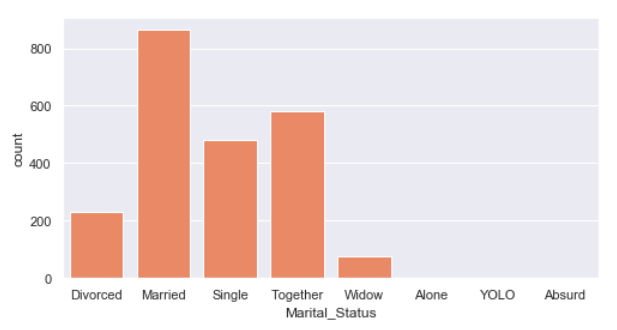
* **Class Weight Attribute**: Some models such as Logistic Regression and Decision Trees have a parameter called **Class\_Weight** which passes to the algorithm the weights associated to each class. This allows us to penalize more the mistakes in the minority class, and value more the correct guesses of the minority class.
* **Over Sampling**: As we can see in [3] is a way to generate synthetic data of the minority class, to increase the ratio between classes. In our case we decided to use Synthetic Minority Over-sampling Technique for Nominal and Continuous also known as **SMOTENC**, which calculates new records based on the distance to a k number of neighbours. We decided to use this one instead of just **SMOTE**, because it can deal with categorical variables, but we also implemented SMOTE for cases in which we only use numerical variables. We also implemented **ADASYN** which uses SMOTE as a base version but adds a random small value to the points, which increases the variance a bit.
* **Proper Algorithms**: There are some algorithms that are more suited to deal with imbalanced cases, such as AdaBoost and Gradient Boosting, because they can focus more on difficult cases. Also, there are some methodologies such as **Stratified K-Fold Cross-Validation**, which help us maintaining the same proportion of 0’s and 1’s, among the different folds, in cases on imbalanced datasets.
* **Different Metrics**: Particularly in these cases, we should look for different measures, because Accuracy and Recall might be misleading. So, we will also use other metrics such as a ratio with the maximum possible profit and F1 Score.

There was also the option of using **Under Sampling**, which is basically reducing the quantity of rows with Response 0, so that their number is as close to the ones with Response 1 as possible. However, we believe that we wouldn’t benefit from reducing our dataset this much, so we decided not to use it.

Then, we were able to identify missing values. The only variable which has **missing values** is the Income (24 values). We figure out two different ways of treating them, either by simply removing the rows that had missing values or impute their values. Different variable types require different ways to impute missing values, here are the ones that we are using:

* For categorical variables we are looking at the mode because it gives us the most frequent category. For numerical variables we decided to use the Kolmogorov-Smirnov normality test that examines if variables are normally distributed. If so, we are using the mean to impute the values, otherwise we are using the median.

We also found that we had several **duplicates**. Some of them with different Responses and others with the same Response. Which in the first case is clearly noise that can have implications when developing our models, thus we decided to eliminate all clients that were duplicates and the only thing that changed was the response. We considered the hypothesis of comparing the ratio between the number with response 0 and response 1 and keep one sample of the group with the most records, however, since the biggest ratio we found between classes was 2:1, we forgot this idea. For the other cases, when the whole line is a duplicate, we just kept one of those. After this duplicate removal, we removed 220 records.

Besides that, one of the first things we saw was the type of variables we had, because different types require different treatments. We had some strings where it should be a date type, so we changed that. Plus, categorical variables (Marital\_Status and Education) might require encoding, so we had a method to do **one hot encoding** to them.

We also saw, by means of a countplot visualization, that in the case of Marital\_Status we had categories with strange values such as Alone, Absurd and YOLO, so we deleted these **categorical outliers** before doing the One Hot Encoding.

Also, for numerical variables we spotted several **outliers** both from univariate and multivariate perspectives which had to be treated during the pre-processing, which means we will have to implement several outlier detection methods on our pipeline to avoid noise on our training data while creating our model. Here are some of the visualizations we use to look for outliers, as well as their techniques:

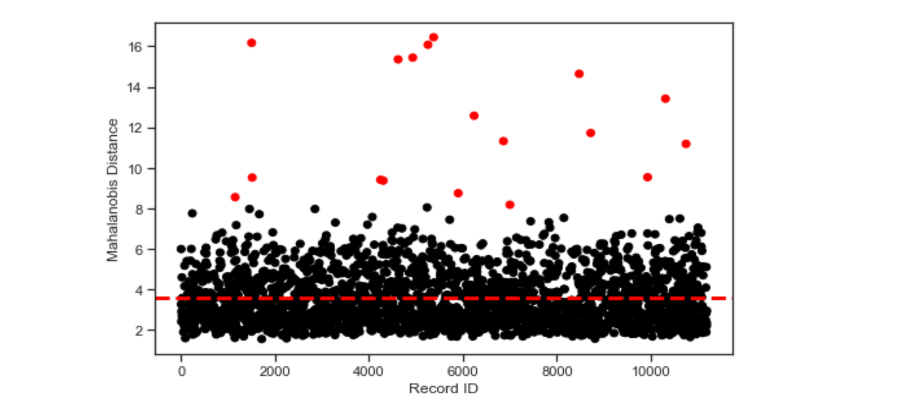
**UNIVARIATE PERSPECTIVE**

Univariate analysis is the simplest form we found to analyse our data, because we look to each variable isolated and we check for outliers in this univariate perspective. There are several ways to see this, here are some of the methods we implemented:

* **Filter By Standard Deviation**: Classifies as outliers all values that are 3 standard deviations above or below the mean.
* **Manual Outlier**: On our first analysis, we looked at the boxplots, at the max and the min values and decided which values should be considered as outliers.
* **Z-Score**: TO DO
* **Boxplot**: TO DO
* **Robust Z-Score**: TO DO

**MULTIVARIATE PERSPECTIVE**

Looking only at the univariate perspective might be misleading because some outliers might only appear when checking multiple variables at the same time. These outlier multivariate techniques can be divided in statistical methods and data-mining related methods. Thus, we also implemented several ways to look at outliers in the multivariate perspective:

* **Isolation Forest**: As said in [4], this method is a different approach from the other ones because it detects anomalies based on the concept of isolation and not on the concept of distances. However, as we can see in [5], Isolation Forests’ decision boundaries are either horizontal or vertical, which lead to a few problems in outlier detection. So, we found that there is another algorithm called **Extended Isolation Forest** that can help us, because it can introduce a slope on the decision boundaries’ lines. This improvement to Isolation Forests is also said to generalize well into higher dimension.
* **Mahalanobis Distance**: Mentioned in [6] as a statistical method for multivariate outlier detection, usually sees which records are far away from the centre of the data distribution. Here is an example of the results on our dataset without any pre-processing, in red we have 18 points that were considered outliers by this distance.
* **DBSCAN**: This is a density-based clustering method that works by grouping together points that are close to each other and marks as outliers the ones that are left alone. In [7], Çelik et all performed an analysis that showed that DBSCAN had several advantages over certain statistical approaches when discovering outliers, so we decided to implement it.
* **Elliptic Envelope**: Another way to check for outliers, here a robust covariance estimate is fitted to the data, and thus an ellipse is fitted to the central data points, ignoring points outside the central mode. Again, the furthest away from the central ellipses, the highest the probability of being outliers.
* **Local Outlier Factor**: This method is also based on the number of neighbours and how isolation a point is. It identifies samples that have a substantially lower density when compared to their neighbours and consider them as outliers.
* **One-Class SVM**: This algorithm tries to adjust to the actual distribution of the data. The big advantage of One-Class SVM is that it can capture the real data structure, but because of the parameters we can also overfit the data.
* **Cooks Distance**: TO DO
* **K-Means: TO DO**

Our idea with all these implementations is to compare between the different results. We believe that points that are consistently called “outliers” will probably be outliers. So, we implemented a Rank that gets the points that are most of the time considered outliers. Let me explain better it’s functioning:

* **Outlier Rank**: After running the several outlier detection techniques from the univariate models we get dictionaries with the ID’s and the variables in which they were considered outliers, and from the multivariate technique we have just their ID. Then this information is joined in a list and if Smoothing is equal to False, all records above a certain threshold are eliminated. On the other hand, if Smoothing is equal to true, the outliers identified in the univariate perspective are Smoothed using the Boxplot and IQR. The ones in the multivariate way are eliminated.

During our analysis we also noticed that our variables are in very different scales, on [8] Jason Brownlee says that large input values lead to large weight values which makes the models unstable and can make them perform worse, therefore we will have to normalize the variables. When considering the several options we have to normalize the variables, we decided to use **MinMaxScaler**, because it puts all our variables in the same range ([0:1]), consistent with some of the variables we already had.

**FEATURE ENGINEERING**

Now that this first analysis is done, we started thinking about ways to create new insights on our data, this can be done in a process called **Feature Engineering**, which involves both creating new variables and ideally selecting the ones that give us more explanation power. Let’s then divide this in two, Feature Extraction and Feature Selection.

**FEATURE EXTRACTION**

The whole point of Feature Extraction is to build derived values, from the existing ones, that are more informative or less redundant, which will help us when building our model. It can also be considered a dimensionality reduction process where the initial set of variables is reduced but can still describe the initial dataset. There are several ways to do Feature Extraction, here are some that we implemented:

* **Business Features**, which basically consists on creating new relevant features based on the existing ones. Here is a table with the features resulting from this transformation and the way they were calculated:

|  |  |  |
| --- | --- | --- |
| New Variable | Formula | Description |
| Total\_Purchases | NumCatalogPurchases + NumStorePurchases + NumWebPurchases | All purchases made by the costumer |
| RatioWebPurchases | NumWebPurchases / Total\_Purchases | Ratio of purchases made in the Web |
| RatioCatalogPurchases | NumCatalogPurchases / Total\_Purchases | Ratio of purchases made by Catalog |
| RatioStorePurchases | NumStorePurchases / Total\_Purchases | Ratio of purchases made in Store |
| Web\_Purchases\_Per\_Visit | NumWebPurchases/ NumWebVisitsMonth | Ratio of purchases by visit on the web |
| Age | Now().Year – Year\_Birth | Customer’s age |
| NumberDaysCustomer | Today’s Date – Dt\_Customer | Number days since the first purchase |
| TotalAcceptedCampaigns | AcceptedCmp1 + AcceptedCmp2 + AcceptedCmp3 + AcceptedCmp4 + AcceptedCmp5 | Total number of accepted campaigns |
| TotalMoneySpent | MntWines + MntFruits + MntMeatProducts + MntFishProducts + MntSweetProducts + MntGoldProds | Sum of money spent on all categories |
| RatioWines | MntWines / TotalMoneySpent | Ratio of money spent on Wines |
| RatioFruits | MntFruits / TotalMoneySpent | Ratio of money spent on Fruits |
| RatioMeatProducts | MntMeatProducts / TotalMoneySpent | Ratio of money spent on Meat Products |
| RatioFishProducts | MntFishProducts / TotalMoneySpent | Ratio of money spent on Fish Products |
| RatioSweetProducts | MntSweetProducts / TotalMoneySpent | Ratio of money spent on Sweet Products |
| RatioGoldProd | MntGoldProds / TotalMoneySpent | Ratio of money spent on Gold Products |
| Income2Years | Income \* 2 | Income of 2 years |
| EffortRate | TotalMoneySpent / Income2Years | How much of the income was spent in store |
| TotalKids | Teenhome + Kidhome | Total number of children |
| Count\_Household | **IF** Marital\_Status = Together or Married  2 + TotalKids  **ELSE** 1 + TotalKids | Number of people per household |
| Income\_Per\_Person | Income2Years / Count\_Household | How much money is available per person in a household. |
| MoneyPerPurchase | TotalMoneySpent / Total\_Purchases | How much money was spent by purchase |

* **Principal Component Analysis**: This transformation changes our initial variables which were correlated between them to a set of values linearly uncorrelated called principal components. The first component will have the largest variance.
* **Linear Discriminant Analysis**: This is a method that tries to find linear combination of features that can characterize or separate two or more classes of objects. This is usually used for dimensionality reduction before classification.
* **Factor Analysis**: It’s similar to PCA, however has a good advantage when compared to it because it can model the variance in every direction of the input space independently.
* **Independent Component Analysis**: Here we are not trying to reduce the dimensionality, what ICA tries to do is to maximize the statistical independence of the estimated components.
* **Box-Cox Transformations**: Several transformations such as logarithms, squares and multiplications that were applied to the existing features to try to extract more meaning and explanation power.
* **Multi-Factor Analysis**: It is an improved version of PCA when we also have qualitative variables, so we thought it could be interesting to see if our results were better with this.

**FEATURE SELECTION**

On the other hand, feature selection will help us selecting the most informative variables. The ones who can explain better the behaviour of our target variable. Also, as said in [9], feature selection is used to avoid the curse of dimensionality, which is a huge machine learning problem. Again, several techniques were used, here is their explanation:

* **Correlations**: Checks the correlation with the target variable. Our idea is that the most correlated a variable is with our target, the best it will explain its behaviour.
* **Linear Regression**: Our idea here is to see which variable by itself can explain better the behaviour of the target value. So, we did regressions of all our variables with the target variable and chose the most capable ones.
* **Fisher Score**: TO DO
* **Information Gain:** Decision Trees chose the variables they use based on Information Gain, which is calculated using Entropy. So, our idea was that these features probably will also be good on our models.
* **Recursive Feature Elimination**: This uses an external estimator, in our case we opted for logistic regression, that assigns weights to features. It recursively eliminates the least important features, to keep just a small amount of the most relevant ones.
* **Select K Best**: Selects features according to the k highest scores of a function that is decided. In our case we used the f\_classif which is a method based on the **F-test** that estimates the degree of linear dependency between variables.
* **Extra Trees Classifier**: This algorithm choses the most important features when building the model, and it returns a parameter called “feature\_importances\_” that has values for each feature, the higher the value the more important the feature, so we will use this list as a feature selection.
* **Genetic Algorithms**: We are using Genetic Algorithms in feature selection. Basically, FeatureSelectionGA algorithm receives as parameter one algorithm and all the variables that can be combined. Then it iterates and returns the best combination of variables it found for that algorithm.

As in the outlier part, we decided to implement a rank here. Basically, we get the variables selected in the different models and we count how many times each of them was selected. We also select a threshold that will define how many variables the rank will retrieve.

The last thing we are using in this Feature Selection is checking if we don’t put highly correlated variables in our models. Because highly correlated variables won’t give us a lot of new information and will increase the dimensionality, which we want to avoid.

**ALGORITHMS**

The last thing to do on our research part, was to try to find algorithms that could help us finding the best model for our problem.

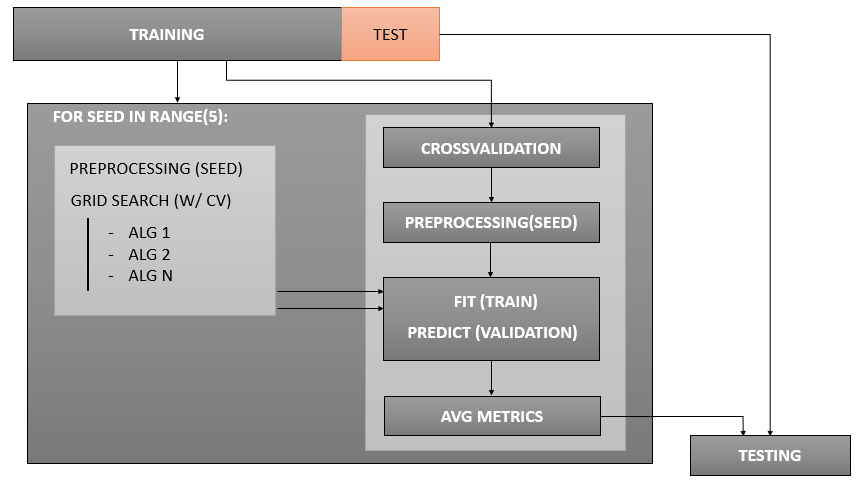
* **Decision Tree**: Non-parametric supervised learning method that learns simple decision rules inferred from the data features.
* **Multi-Layer Perceptron**:  A class of feedforward artificial neural network.
* **Logistic Regression**: The probabilities describing the possible outcomes of a single trial are modelled using a logistic function.
* **Voting Classifier**: We are using this this type of **Essemble** to combine different algorithms and use a majority vote.
* **AdaBoost**: This algorithm begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases
* **Gradient Boost**: This algorithm produces a prediction model in the form of an ensemble of weak prediction models, such as DTs. It has some advantages because can handle data of mixed type and has a good predictive power.
* **XG Boost**:
* **Extra Tree Classifier**: It fits a number of randomized decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
* **Naïve Bayes**: We used ComplementNB which is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets.
* **SVM**:

**OPTIMIZATION ALGORITHMS:**

* **GRID SEARCH:**
* **BAYESIAN OPTIMIZATION:**

After this initial exploration and research, we decided to construct our pipeline on Pycharm and we passed the things of Jupyter Notebook to Pycharm, so that we could start combining our multiple options.

**PYCHARM**

The first thing we had to do here was to agree on a Pipeline, i.e., a way to organize our code and our logic. Here is a picture of our Pipeline:

We decided to perform a first split on our data that resulted on a Training Set with (80%) of data and in a Testing Set with (20%). Then with those 80% of Training we passed it to a loop to run the 5 different seeds. Then, inside this loop we have two parts that run in parallel, the first one (on the left) in which we do a pre-processing and then run an optimization algorithm with cross-validation on all the algorithms that allow it, to try to look for good parameters for our model. Because we believe we can benefit form this information when we will actually do the whole process. Then, we have the second part in which we start by applying the **Stratified K-Fold Cross-validation**, which is more suited when we have these imbalanced datasets. Then we do the selected pre-processing with the same seed as before. And we fit our models, with the training data and we try to start them in a better position because of the information gained on the first part. Then we predict on the validation part and we get the metrics for this part. After the whole loop, we will see which models are consistently better and we will apply them to the testing dataset that was separated in the very beginning. We are doing this because as we can find in [10] and in several other literature, is recommended that we have a testing set that is not used on cross-validation to access the expected performance of our model when facing new data.

Using the pipeline described above, we decided to do run a few baselines that gathered some of the methods we implemented, to see how things work together and from which methods we can benefit more.

|  |
| --- |
| **BV0** |
| Cross-Validation |
| Generate Dummies (training join with validation) |
| Gridsearch |
| Drop missing values (training, validation) |
| Boxplot outlier removal percent= 0.03, 1.5(training) |
| Normalize (Min-max Scaler) |
| Extract Business features |

|  |
| --- |
| **BV1** |
| Cross-Validation (Stratify k-folds) |
| Generate Dummies (training join with validation) |
| Gridsearch |
| Impute missing values |
| Mahalanobis |
| Normalize (Min-max Scaler) |
| Balancing do dataset (weights) - SmoteNC |
| Extract Business features |
| PCA |
| Correlation based feature selection |

REFERENCIAS

[1] <https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/>

[2] <https://towardsdatascience.com/comparing-different-classification-machine-learning-models-for-an-imbalanced-dataset-fdae1af3677f>

[3] <https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html#smote-adasyn>

[4] Fei Tony Liu, Kai Ming Ting, Zhi-Hua Zhou, Isolation-Based Anomaly Detection, ACM Transactions on Knowledge Discovery from Data (TKDD), v.6 n.1, p.1-39, March 2012

[5] <https://towardsdatascience.com/outlier-detection-with-extended-isolation-forest-1e248a3fe97b>

[6] Ben-Gal, I. (2005). Outlier detection. In Data mining and knowledge discovery handbook (pp. 131-146). Springer, Boston, MA.

[7] M. Çelik, F. Dadaşer-Çelik and A. Ş. Dokuz, "Anomaly detection in temperature data using DBSCAN algorithm," 2011 International Symposium on Innovations in Intelligent Systems and Applications, Istanbul, 2011, pp. 91-95.

[8] <https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/>

[9] Mladenić D. (2006) Feature Selection for Dimensionality Reduction. In: Saunders C., Grobelnik M., Gunn S., Shawe-Taylor J. (eds) Subspace, Latent Structure and Feature Selection. SLSFS 2005. Lecture Notes in Computer Science, vol 3940. Springer, Berlin, Heidelberg

[10] <https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7>