Master Degree in Data Science and Advanced Analytics

Major in Data Science

TechScape E-commerce

Machine Learning

Teaching Staff:

Roberto Henriques
Carina Albuquerque
Lara Oliveira

Group 35:

Filipa Alves, m20210662
Helena Oliveira, r20181121
J. Daniel Conde, m20210656
Leonardo de Figueiredo, m20210667
Leonor Candeias, m20210990

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

December 2021

Contents

Abstract

With the intention of classifying potential purchases based on their Google Analytics metrics, our project centred itself on creating a predictive model that achieved this with as high an F1-score as possible on the test dataset. We began by completing the requisite data exploration and pre-processing, in which we conducted coherence checks, data splitting, missing values treatment amongst other techniques. We concluded by conducting various feature selection methods as well as oversampled our dataset as it was moderately imbalanced. Our team utilized numerous classifiers; all having been tested with various combinations of datasets using processed and oversampled data as well as data including outliers. The classifier that yielded the best results was the stacking classifier with an F1-score of 0.671 composed of the Neural Network, Logistic Regression, SVM and Random Forest.

Business Needs

In the year 2020, the world is not dissimilar to that envisioned by Gordon Moore, smartphones, computers, and screens have become ubiquitous, with most users spending hours per day behind some form of computer. Stemming from the growing necessity for many tech consumers to reconnect with the physical world, the company TechScape was established to provide their customers with goods and services associated with digital detox.

As the team of data scientists hired, we have been tasked with the creation of a predictive model capable of predicting the probability of any given prospective customer converting to a sale, through the company's online sales channel. Our team aims to achieve this by way of applying several supervised learning techniques to a dataset comprising of an initial 9999 records, each containing standard Google Analytics metrics as features, with the target variable being whether a prospective customer converted to a sale, designated by the variable "Buy". Our aim to achieve the highest F1-score possible, testing our results against a ground truth set of unseen records.

Data Exploration

The first approach to the given data was to analyse all features and their correspondent values within the data frame created. It is relevant to know if there were any missing values and all data types we were going to deal with in this data analysis. With this in mind, the number of observations/records and the number of missing values were checked.

After that, we made use of the method *describe()* to check the mean, maximum and minimum values of all features. We proceeded with the data preparation, meaning that a few changes were made to our initial data set. Each record became represented by the Access ID making use of these unique values that each customer uses as an identifier. Types of feature fields were then changed to more appropriate type objects with the main purpose of usefulness: datetime, string, int32 and float32.

Following this, a heatmap as well as a pairwise relationship graphs were created to demonstrate all variables correlations and, more specifically, the impact of the variable '*Buy'*. The spearman correlation matrix (Fig 1 in the Annexes), displays a significant positive correlation between the variables '*Buy'* and '*GoogleAnalytics_PageValue'* as well as between '*GoogleAnalytics_BounceRate'* and a negative correlation with '*GoogleAnalytics_ExitRate'*. Another aspect about the correlation matrix is the

significant correlation between the variables '*GoogleAnalytics_ExitRate'* and '*GoogleAnalytics_BounceRate'.*

In the pairwise relationship graph (Fig 2 in the Annexes), the variable '*Buy'* also seems to have a greater relevance and correlation when compared with the '*GoogleAnalytics_PageValue'* variable.

Coherence Check

The data was checked for incoherent information given the problem statement and context to ensure the validity of the analysis. The rules checked are present in Table 1 of the Annexes.

2,98% of the dataset was incoherent according to rule 2. Given that almost 3% is still a large percentage of the data, and to avoid losing any important information that could be present in those records, those values were replaced. This means that, in records where the amount of time spent in a page (account management, FAQ or products and services pages) is zero and the number of pages visited is greater than zero, the number of pages visited was replaced by zero. This way no information is lost due to a possible inconsistency in the data collection.

Split the data

Subsequently, we proceeded with the data partition into a training set to train the model and a validation set to evaluate the trained model and control overfitting. 70% of the data was used for training, the remaining for validation in order to tune the hyperparameters and control overfitting. This partition was done before pre-processing to assure the validation set does not contain any information from the training dataset so that when model performance is assessed on the validation set, the results are as close as possible from a future unseen data. By specifying the stratify parameter, the proportion of the classes of the target binary variable is the same on both new datasets, crucial for imbalanced datasets. 70% of the data are then used for training the models and 30%. In order to avoid any patterns due to the ordering of the records the data was also shuffled before the partition.

Since there were both categorical and numeric variables, a separation between categorical and numerical variables was made filtering these subsets with a list previously created using only categorical variables.

Data pre-processing

Outliers Treatment

Manual Limitation

The first step to the treatment of outliers consisted of the most direct approach – manual limitation. If a datapoint's behaviour seemed to be too detached from the norm within a certain dimension, it would be considered an outlier. Column plots and box-and-whiskers plots were created to evaluate the distribution of the variables, and a function was created to exclude the selected points from the training dataset. The criteria within the function were manually imputed (Fig 3, 4 and 5 in Annexes).

It is important to mention that this analysis was done exclusively for the training dataset. To evaluate the model and avoid overfitting, the validation dataset should be as untreated as the testing dataset. For this reason, this dataset was not screened for outliers.

IQR

As the IQR method is only applied to normally distributed data, this was not an option. Despite that, more complex algorithms were not considered since we wanted to have control on how the records were filtered once in machine learning field this is a very sensitive issue.

2-dimensional outliers

After the training dataset was swept of one-dimensional outliers, it needed to be checked for outliers across two dimensions. The chosen method to carry this out was the visualization of pairwise relationships between variables, followed by the human selection of outliers based on datapoint behaviour.(Fig 6 in the Annexes)

Missing Values

We verified that there were no missing values among all records, having also searched for values such as _, ' ' and others that sometimes appear to identify a missing value.

Scale the data

In order to have some insights on what scaler is most appropriate for the dataset without outliers, we ran an algorithm overly sensitive to the scaling, KNN, with 5 neighbours and a set of 5 metric features, to check which performed better: *MinMax*(0,1) or *MinMax*(-1,1). In fact, that both scales performed remarkably similar (Fig 7 in the Annexes) . Thus, we proceeded with the data scaled between 0 and 1. The robust scaler was not tested since the outliers were already removed and the standard scaler was not that appropriate since it is used on normal distributions, which is not exactly the case of our variables.

Furthermore, a data frame with outliers was also created in order to further test if the models may work better with the presence of extreme values. Central tendency measures are usually highly impacted by outliers. To avoid this, the robust scaler was used to scale the data. It is important to note that, in order to avoid data leakage, the scaler fitted to the training dataset was used to scale data in both validation and test sets.

Feature Engineering

In order to find the best prediction for our target variable (*Buy*), it was decided to create several variables that could have a greater explanatory impact on the target (Table 2 in the Annexes).

One Hot Encoding

One hot encoding was applied to the categorical variables since they are not ordinal, otherwise Label encoding would have been more appropriate. This was used since some algorithms do not operate with nominal variables, so that binary ones are preferred.

Redo outliers check

A careful analysis regarding the outliers was conducted, since after the creation of new variables, some outliers could have arisen. In fact, no clear extreme values were found. (Figure 8 in Annexes).

Feature Selection

To access the relevancy of our binary variables the variance threshold was used, the chi-squared test and decision tree. All the other methods mentioned above were used to check metric features importance to predict the target.

Regarding the univariate analysis of the variables, we concluded none of the metric features were univariate. On the set of binary variables 16 binary variables (Table 4 in the Annexes) had a considerable variance ($0.8 * (1 - 0.8) = 0.16$). By applying this computation, only binary variables that are either one or zero in less than 80% of the samples were partially kept, thus accounting for the variability of our dummy variables.

Filter Methods

The **Chi-squared** test was also a valuable tool since it basically performed a significance test. The **correlation matrix** computed with the spearman coefficient (we have no evidence of linear relationships to consider the Pearson coefficient) was carefully analysed to make sure each set of variables did not have any significant correlations among its features, otherwise redundancy would arise (Fig 9 see in Annexes). The **ANOVA test** implementation was done through the *f_classif()* function from *sklearn* library. To select the best metric variables, the ones with largest values were selected, *SelectKBest* class was used.

Wrapper Methods

One of the feature selection algorithms implemented was an RFE, with a fivefold embedded cross validation through *sklearn.feature_selection* **RFECV** (Fig 10 in the Annexes). It was decided to incorporate this technique to reduce the number of irrelevant features, which not only leads to an increase in efficiency but allows our model to be more effective as well. **Forward Sequential Feature Selection** was implemented through *SequentialFeatureSelector* method by applying a Ridge Regression. Backward Sequential Feature elimination was also tried, however as the results were very similar to the previous method, the former was chosen since we had to test a large number of variables and thus the problem of multicollinearity arises.

Embedded Methods

The **Ridge Regression** was also used, as well has a Decision tree (the upper variables chosen to prune the data are more relevant accordingly to the criteria used by gini) – Fig 11 in the Annexes.

As these methods produced different results (Table 5 in the Annexes) regarding the importance given to each variable, we considered the features most frequently chosen by the methods. We got 6 sets of variables (Table 3 on the Annexes). It is important to note that '*log_Page_Value*' is present in all sets of features since it is the only variable reasonably correlated with the target (spearman correlation=0.63).

Check for an imbalanced dataset

SMOTENC - Synthetic Minority Over-sampling Technique for Nominal and Continuous

Since the training data is mildly imbalanced, there is a risk that the models applied will learn very well how to predict the majority class (0), however as there are very few examples of the minority class (1),

this will be poorly predicted. Thus, the accuracy score would not be trustworthy to assess the real performance of the model, since it is trivial to get a high accuracy score by simply classifying most observations as the majority class.

To address this issue, we oversampled the training data (minority class) resulting we records for each class using SMOTENC. This technique is a variation of SMOTE that can deal with categorical features (the set of variables chosen had also binary variables). The algorithms were performed for both the oversampling training dataset and for the unbalanced dataset.

Modelling

Training the models in the oversampled data did not produce enthusiastic results since it did overfit loads (the F1 score from the training dataset is greater than from the validation set, leading to poorly predictions on unseen data). This issue was attempted to be solved by changing the number of features or reduce the algorithm complexity, but it could not be fixed. Therefore, this data was not further used to build the models.

Since the dataset is moderately imbalanced, we tried to give that information to the algorithm by specifying *class_weight = 'balanced'*, giving focus to the minority class by using an inverse weighting from the training dataset. So higher class-weight means the class gains importance (its errors are considered more costly than those of the other class). As class 0 is 5.6 times more frequent than class 1 in the training dataset, we would specified *class_weight = {0:1,1:5.6},* but for the sake of robustness we specified *class_weight='balanced'*, which automatically assigns the class weights inversely proportional to respective frequencies of the classes, wherever possible. We found a decay in precision, since more records were predicted as class 1, on the contrary, recall (True positive rate) improved.

The **Decision Tree** model was optimized for various values for the minimum samples to split, which is an especially important parameter to avoid the tree to grow too much and thus overfit. The two criteria to access the better feature to split were remarkably similar so we picked 'gini' since it is less time consuming. The class weight parameter was also defined as explained above, as well as the splitter as 'random' to choose the best random split. We found that using splitter as 'best' would lead to overfitting. The best set of features was found to be *set3* for the model trained with the dataset without outliers. As expected, the difference on performance between the data with and without outliers was very similar since Decision trees is an algorithm quite insensitive to the presence of extreme values, nevertheless for the decision tree model, the dataset without outliers performed slightly better.

The **Neural Network** applied was MLPClassifier. The first step taken was to evaluate the feature set that would result in better predictions while avoiding overfitting. This approach was taken with both regular and oversampled datasets, the former producing the better results. A GridSearch was applied to each instance of the neural network, and a regression fitted to the corresponding dataset. The layer sizes with better results were 3 layers each with (2/3 * number of features) neurons.

The model was also trained with a dataset with no outlier treatment. However, its performance on new data did not improve as expected.

To use **Logistic Regression Classifier**, firstly a Grid Search was applied in order to optimize the hyperparameters (*solver, penalty*, and *C*). Again, *class_weight = 'balanced'* was used and the best set of variables proved to be *set_new2*. After found the optimal parameters, several thresholds were tested in the Classifier and the best one proved to be 0.55 (Fig 13 in the Annexes). This process was repeated using the dataset with outliers, where the best threshold was 0.66 (Fig 12 in the Annexes). The results in both datasets weren't significantly different, even though the dataset without outliers performed slightly better.

When implementing the **KNN** model, we ran a grid search on both oversampled and non-oversampled feature sets. We then applied the optimal algorithm, *leaf_size, metric*, and *weight*s to models with a varying values of K (Fig 14 in the Annexes), and for each K predicted and scored against both the training and validation sets. Visualising the results allowed us to choose an optimal K for our final model. We only considered the set of variables with metric features, *set1_no_bin*, due to the calculation of distances.

In the **Support Vector Machines** Classifier, a grid search was performed to find the optimal parameters in both the dataset with outliers and without. In the first set the best kernel proved to be sigmoid and in the second one the Gaussian Radial Basis Function (RBF) performed better, which makes sense since the explanatory variables seem not to have a linear relationship with the target. The variable set used in this classifier was the one with only metric features, *set1_no_bin*.

Ensemble Methods

In **Random Forests** we performed a grid search on the maximum features to look for on each split, the number of decision tree estimators, minimum number of samples to split, and the proportion of the sample evaluated. The *class_weight* parameter was defined as *'balanced_subsample'* since as it considers different subsamples (due to bootstrapping), they may have different proportions between the classes, thus for each subsample the proportion of classes is computed. The features used were all features from *set_new, set_new2* and *set1_no_bin*.

Surprisingly, the model with lower *oobscore* was with no restriction on the maximum features to consider for the split, which is more usual on the bagging algorithm. By analysing the plot made, it was possible to notice the number of estimators was almost irrelevant since the line with the lowest error was a straight horizontal, therefore it was run with 20 estimators. After checking the scores from models with different sample sizes, the conclusion was that the best was giving for each decision tree only 60% of the data with bootstrap sample. It can be inferred that basically a *BaggingClassifier* is being used, since the major differences that distinguishes these two algorithms is the lower number of estimators, the size of the bootstrap sample is the same as the imputed dataset and the use of all features on the Bagging. A logistic regression was also applied instead of a decision tree, however the performance was not better. Since without pruning the decision trees, random forest was giving a lot of overfit we had to specify this parameter.

For the **Adaboost Classifier**, the first step was testing all the individual hyperparameters alone (base estimator, learning rate, number of estimators, and algorithm), in order to get a better perspective about them. These results were used after to run a more precise grid search for both the dataset with and without outliers. The base estimator was only tested with Logistic Regression and Decision Trees, which

are the weak learners that better perform in this model, and in both grid searches Decision Trees was the best base estimator. The best variable set proved to be set_new1 and the best results were with the outlier's dataset. This result wasn't quite the expected, as it is known that Adaboost model is sensible to outliers, due to its exponential loss function. The fact that the dataset with outliers is the one that performs better might be explained by the use of an already robust scaled dataset or by the possibility that the outliers' values aren't that extreme.

To optimise the parameters of the **Gradient Boosting** Classifier a grid search was run, however, to search through combinations of parameters more efficiently, classification scores for different numbers of samples, number of features, learning rates, learning rates and max depths were conducted. Grid searches for three different datasets were run and the best results were achieved using the *gradientbooster_out* model with the *set_new1* feature set. This combination then had its predictions threshold adjusted.

To apply the **Voting Classifier** the best 3 models using the outliers' dataset were chosen: Random Forest, Adaboost and Gradient Boost. The best variable set was *set_new1* and soft version of Voting model was applied.

While pursuing the best performing **Stacking** model, various stacking options were created. Considering the previously created models and their optimal feature sets, six options originated. They varied on feature type – continuous vs discrete – and on feature length – shorter lists vs longer lists. The models were stacked by avoiding the use of similar models (i.e. Decision Trees and Random Forests together). Some of our best results were achieved using this method. While combining simple predictors the best results came from the junction of a Neural Network, Logistic Regression, SVC and Random Forest. However, the best stacked model featured the stacking of three other ensemble models – GradientBoosting, Random Forests and AdaBoost.

Performance Assessment

A repeated stratified k fold was applied to the data. The number of folds and repetitions was based on the scarcity of data, and the need to train the model on sufficient data on each fold. The metrics calculated when running *Kfold* were the F1 score (Fig 16 in the Annexes) and *Precision recall* curve was also plotted since considering that our dataset is imbalanced, this metrics account for this fact.

Instead of plotting the roc AUC curve, a more appropriate visualization is the Precision-Recall curve which summarizes how the recall and precision vary as the probability threshold is changed. The threshold is the value above which an observation is considered in class 1. In fact, this change was performed in some of the models developed in this project. (Fig 15 in Annexes). Also, we did check the overfitting for every model by comparing the metrics on the training and validation datasets.

By analysing the Tables 6 and 7 in the Annexes, we were not expecting having algorithms that are sensitive to the presence of outliers (like Logistic regression, Adaboost or Neural networks) perform better with outliers. We can then conclude the robust scaling does play a good part to soften the impact of these observations. In fact, the test data is very likely to also have outliers and it is important the models are trained to somehow account for this matter.

Conclusion

The problem statement posed the question of consumer behaviour. At a time when online sales are growing in an unforeseen manner, websites need to be built to enhance sales as much as possible. To that effect, the habits of a possible costumer need to be modelled.

The models that did this to greater effect were the Stacking Model, The Gradient Boosting and the Logistic Regression, each with the parameters described in previous chapters. They had the F1 scores of 0.671, 0.670, 0.669 respectively. Contrarily to what we had anticipated, the ensemble models did not outperform simple classifiers by a large margin. Perhaps the classifiers we built did not differ too much between themselves either, which could have led to similar inputs to the ensembles. As we know, the lack of differing predictions is weakening to the performance of some ensemble models.

The question remained of what model to choose. We chose to not simply select the model that performed the best on *Kaggle*, as this could result in overfitting to that dataset. We elected to select the model that performed the best according to our evaluation – the repeated stratified K-Fold F1 score (see previous chapter).

However, where this metric is practically tied, business acumen is necessary to make the final call. A Data Scientists job is not only to understand models and crunch data, but also to translate the insights seamlessly into the business context. What this means in this case, is that the task consists of identifying possible costumers. How can the business sell more; which visits are more likely to materialize, who can the business focus their efforts on?

When the problem is seen through this lens, it becomes easier to decide. The correct model to choose is the one that also identifies the most potential customers among the visitors. The model that, while maintaining the same level of overall performance (F1 score) augments the pool of people the business can sell their products to. Therefore, we would rather have false positives than false negatives. The tie breaker criteria for similar models is therefore the Recall.

If a model is amazing at identifying which visitors will surely make a purchase, its prediction errors will naturally be people who could be interested but were classified the opposite way. However, a model that classifies visitor's behaviour and draws a line to only rule out truly uninterested visitors makes sure the digital-marketing budget is being spent the right way.

Among our best performing models, the Stacking Classifier had the highest accuracy in identifying non-costumers (Table 8 in the Annexes). The people the business truly does not want to market products to. That is why we chose it as the final model, and the final predictor of costumer behaviour.

Among 2300 visitors, our final model predicted 409 purchases.

Theoretical Model Explanation

Chi-Square

It is performed to test the independence of two features. Therefore, when a pair of variables is highly dependent, then the chi-squared value is also high, the variable in question can explain the target variable. We conducted this test with the categorical variables and the target and considered keeping the variables with highest dependence.

The **ANOVA test** is an acronym for 'analysis of variance' that determines if two or more samples come from the same distribution or not, using the variance in order to compare the features. This is used between a categorical variable (in our case 'Buy') and a numeric one. Therefore, if the features came from the same distribution, then they are correlated. If this is the case, the numeric variable is accounted for the prediction of the target.

Forward Sequential Feature Selection finds the best feature to predict the target based on the cross-validation score of the algorithm chosen (in our case **Lasso Regression**) and then adds recursively features to form a feature subset. It is a greedy algorithm since it makes the best option at each stage, so it does not have the capacity to see the 'big picture' which might be a disadvantage as well as the necessity to identify how many features to select.

Ridge Regression is an embedded method, which is, the feature selection algorithm is integrated as part of the learning algorithm, combining the pros of filter and wrapper methods. It uses the use L2 regularization (adds a term with the squared value of the weight) being therefore a conservative method since it reduces the impact of the predictors.

Lasso Regression stands for Least Absolute Shrinkage and Selection Operator and used L1 regularization that consists on adding a term with the absolute value of the weight. Both L1 and L2 regularization works to avoid overfitting by its conservative estimation of the parameters.

The **Voting Classifier** is an ensemble model that simply takes into account the findings of each classifier specified, making predictions based on either Hard Voting or Soft Voting. The former will take as the final predictions, the majority prediction from the individual models. In soft voting, the output class is the prediction based on the highest probability averaged of each class by each classifier.

References

Mukherjee, M., & Khushi, M. (2021). SMOTE-ENC: *A novel SMOTE-based method to generate synthetic data for nominal and continuous features. Applied System Innovation*, 4(1), 18.

Websites:

- Site towardsdatascience.com, article "Ensemble Methods bagging, boosting and stacking" visited on the 23rd of November of 2021
- Site stats.stackexchange.com, article "Bagging, boosting and stacking in machine learning" visited on the 23rd of November of 2021
- Site scikit-learn.org, article "Feature Selection" visited on the 30th of October of 2021
- Site stats.stackexchange.com, article "Bagging, boosting and stacking in machine learning" visited on the 23rd of November of 2021
- Site stats.stackexchange.com, article "Is There any Formal Explanation for the Sensitivity of Adaboost to Outliers" visited on the 23rd of December of 2021
- Site medium.com, article "What is One Hot Encoding and How to Do It" visited on the 21st of December of 2021
- Site datascience.stackexchange.com, article "Is There any Formal Explanation for the Sensitivity od Adaboost to Outliers" visited on the 11st of December of 2021
- Site www.svds.com, article "Learning Imbalanced Classes" visited on the 11st of November of 2021
- Site machinelearningmastery.com, article "Learning Imbalanced Classes" visited on the 11st of November of 2021
- Site www.svds.com, article "ROC Curves and Precision Recall Curves for Imbalanced Classification" visited on the 30th of November of 2021

Annexes

Figures

- Exploration Visualizations

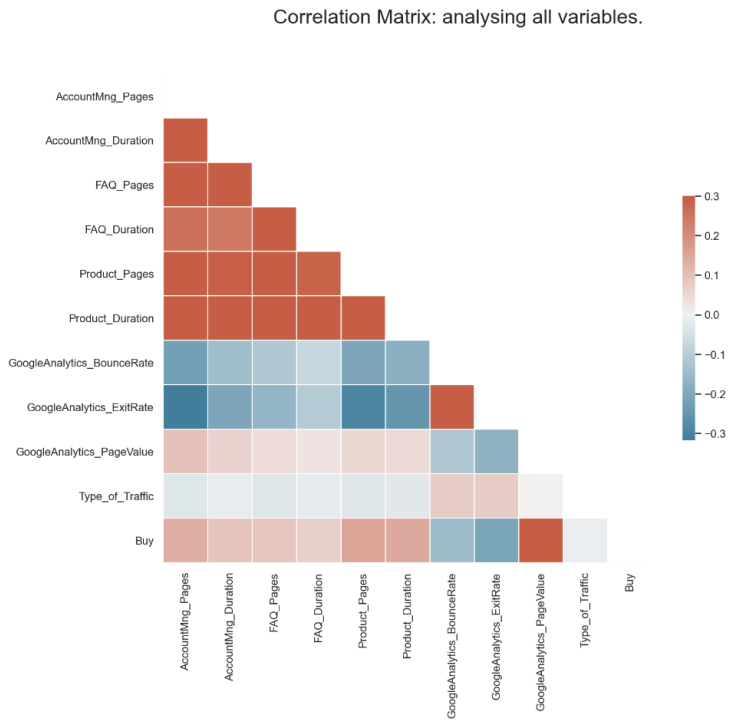

Figure 1: Spearman correlation matrix for the raw data

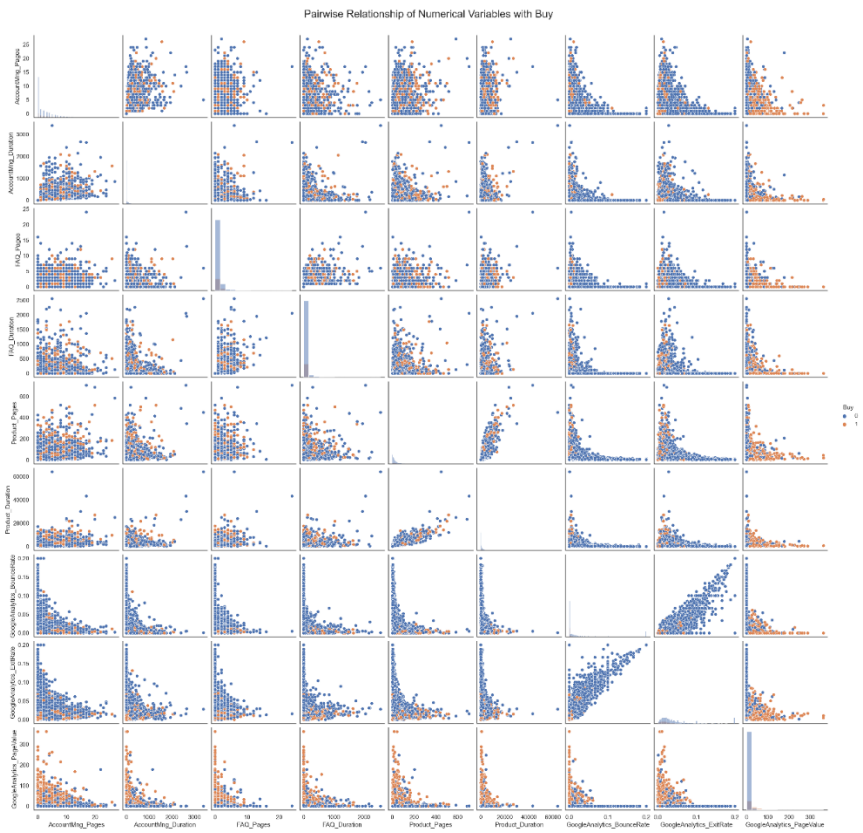

Figure 2: Pairwise relationship of Numerical Variables with '*Buy*'

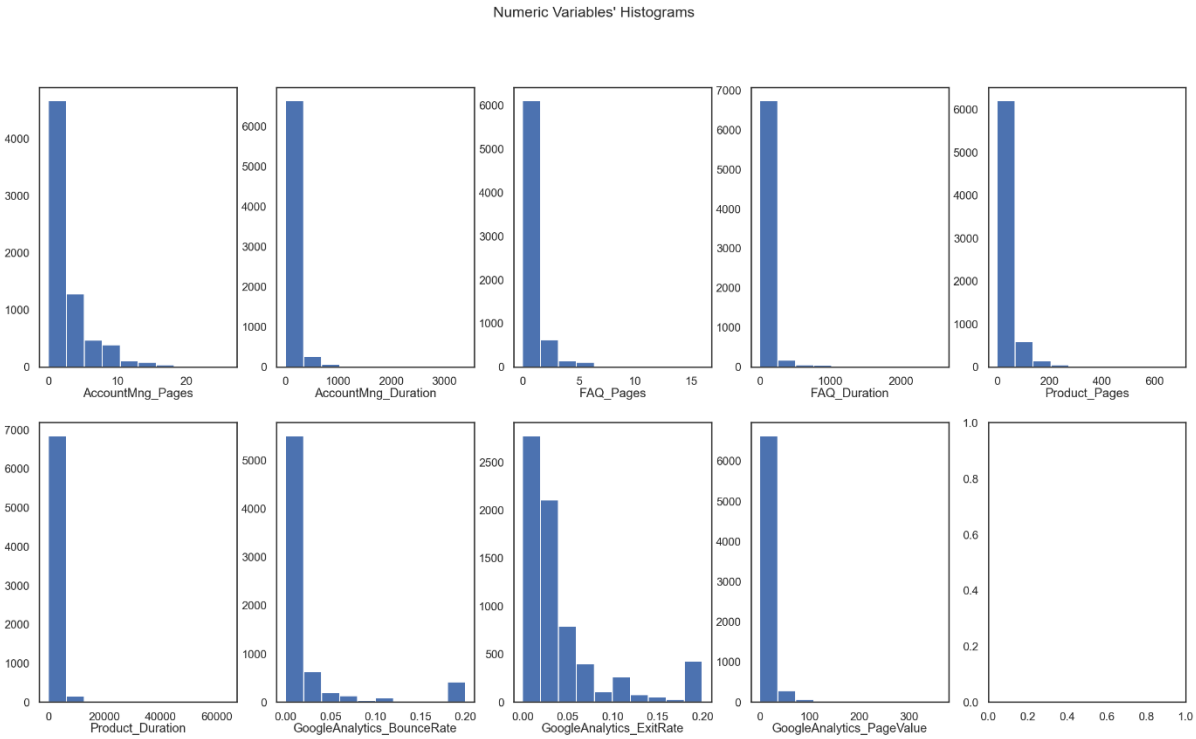

Figure 3: Data before outlier treatment

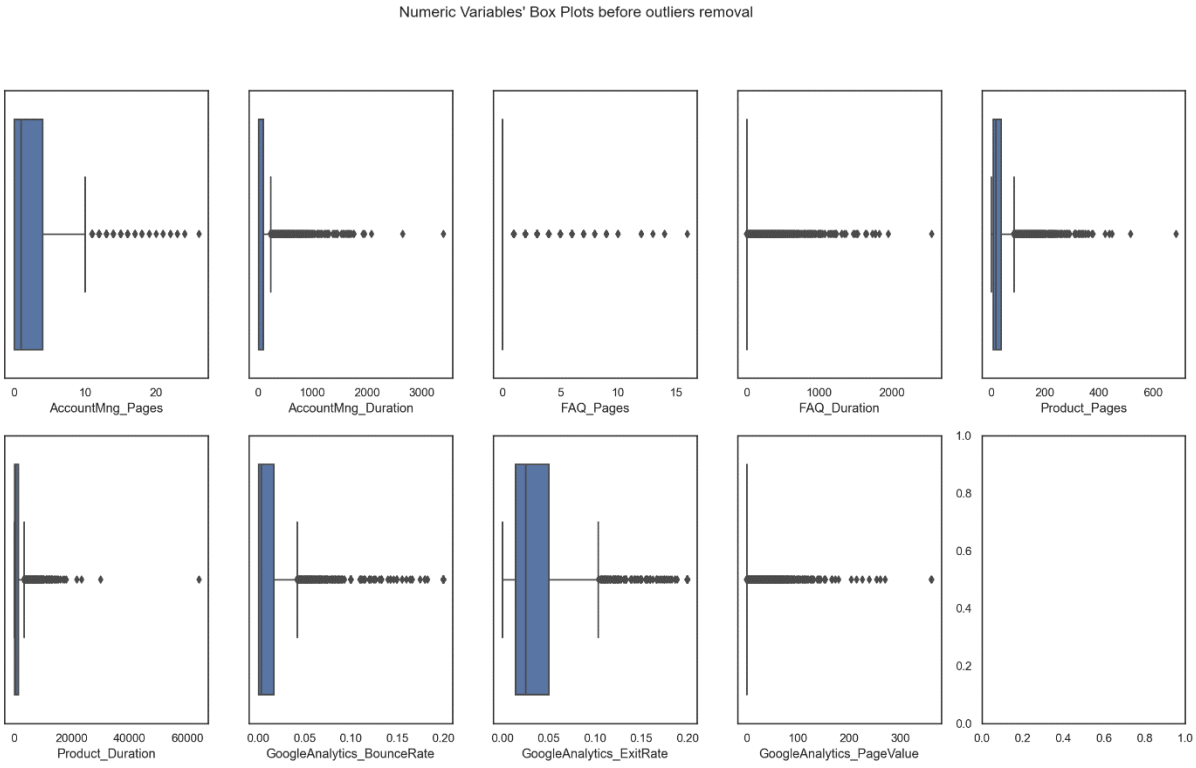

Figure 4: Box plot before outlier treatment

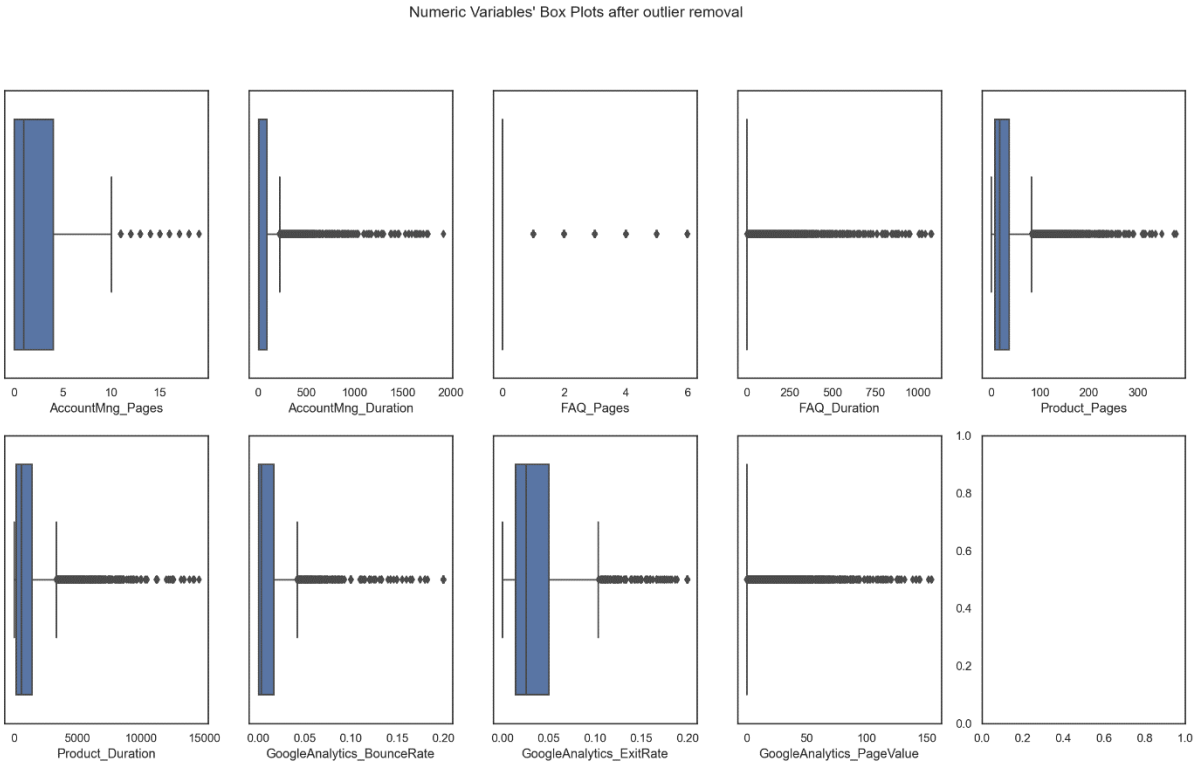

Figure 5: Box plot after outlier treatment

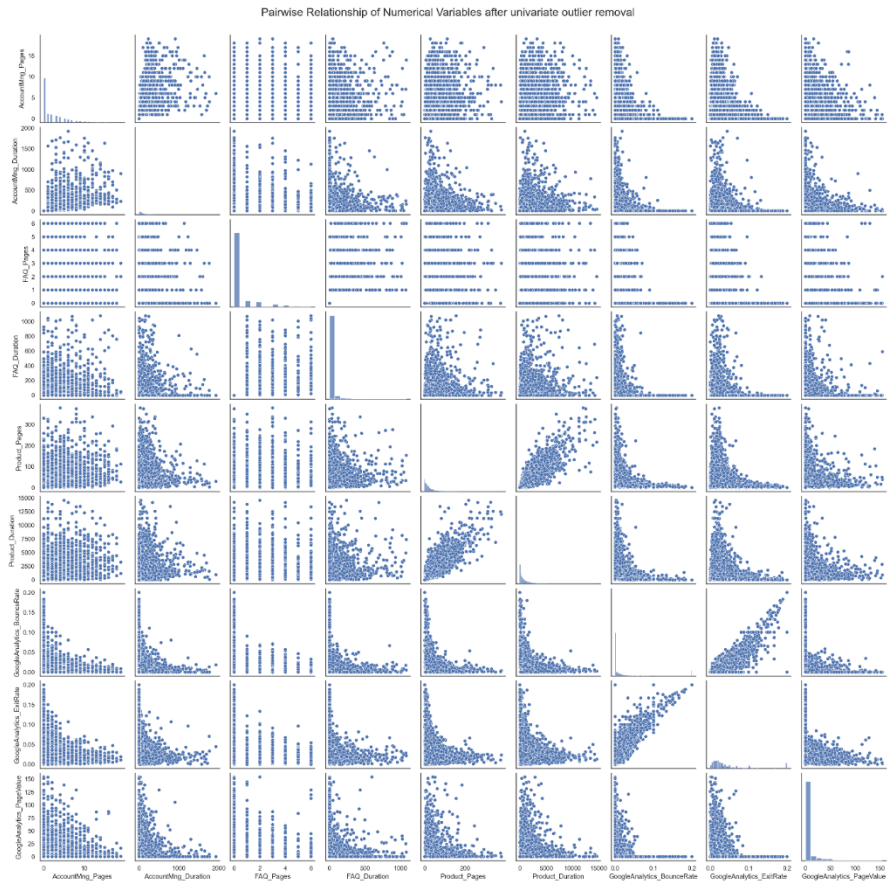

Figure 6: Multidimensional outliers

- Data pre-processing

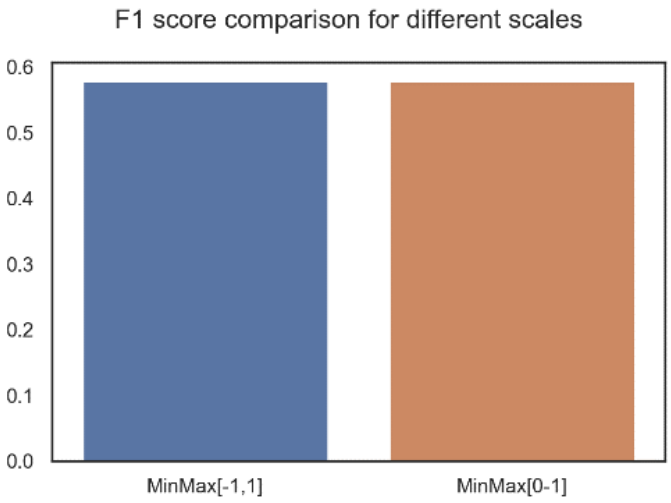

Figure 7: Scales comparison

- Feature Engineering

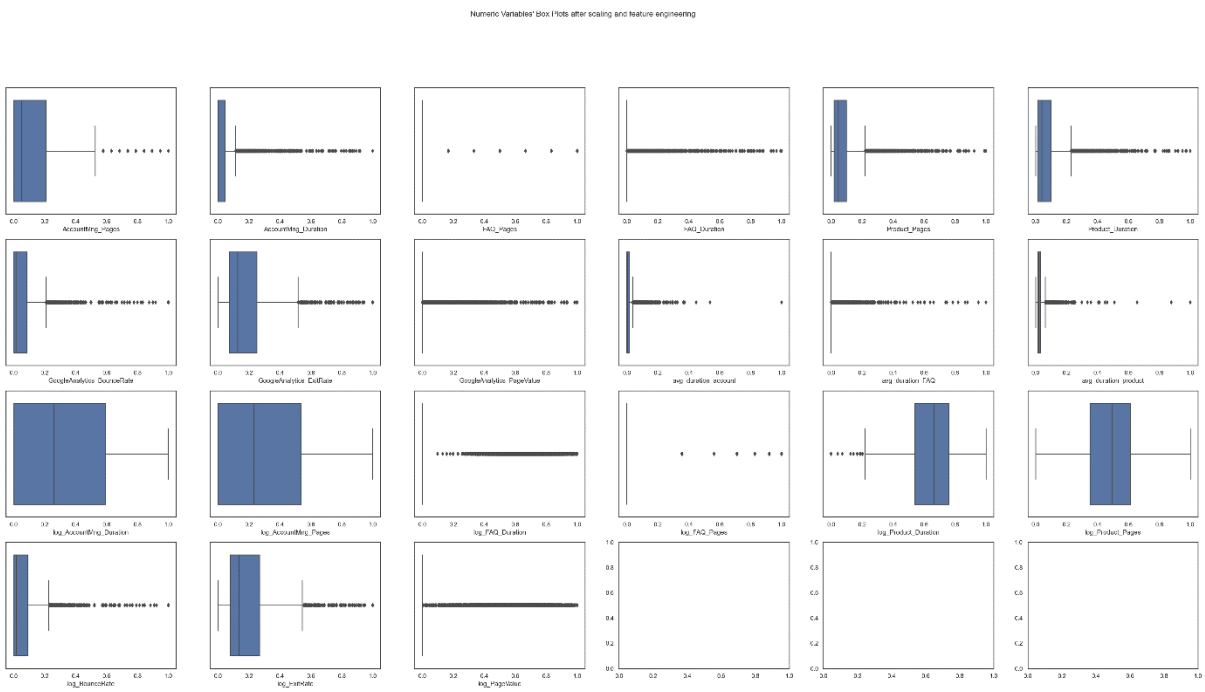

Figure 8: Box plot analysis, after feature engineering

- Feature Selection

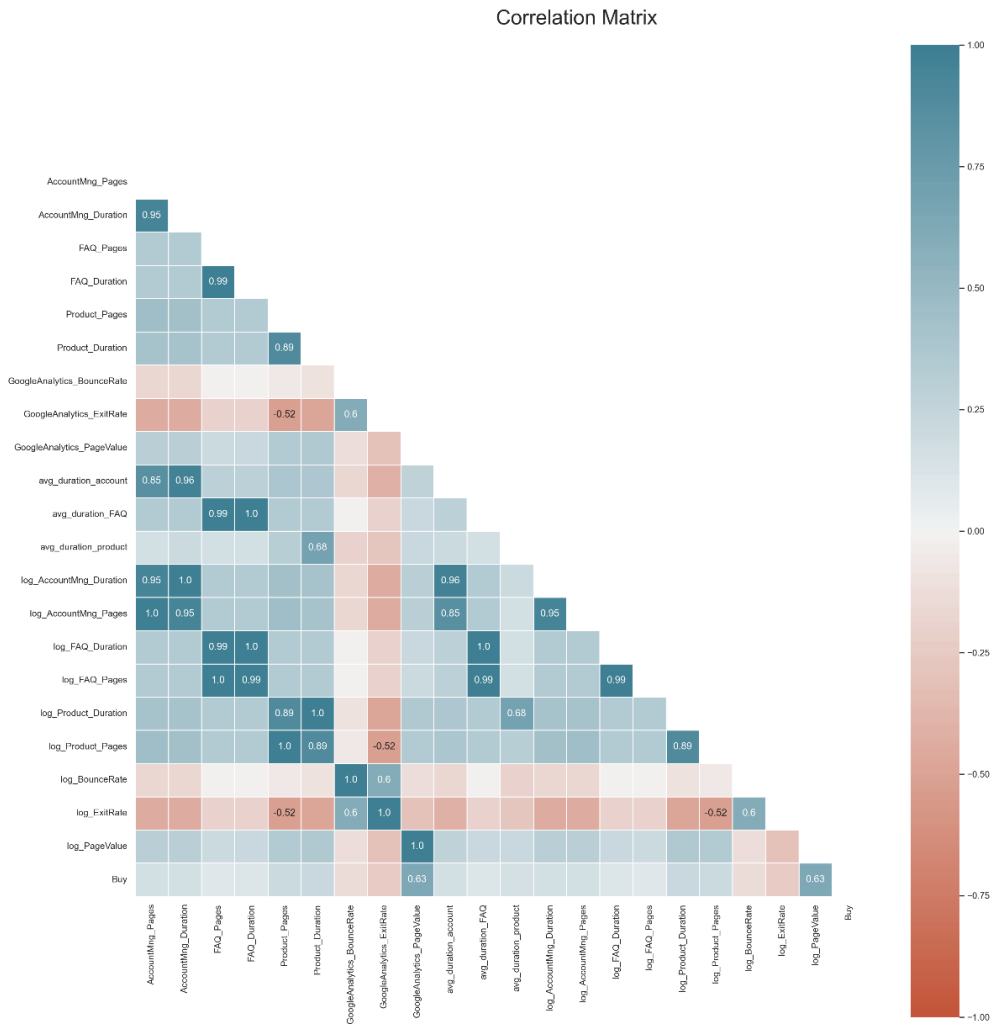

Figure 9: Spearman correlation matrix for the data after creating variables

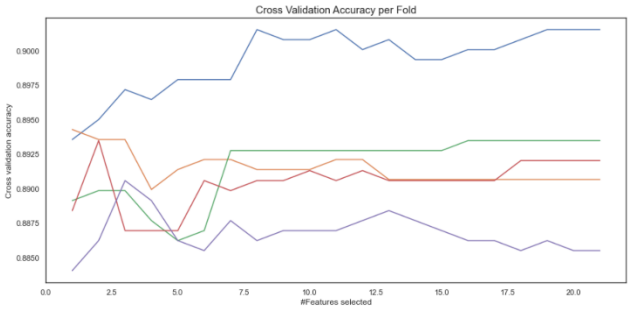

Figure 10: RFE k-folds

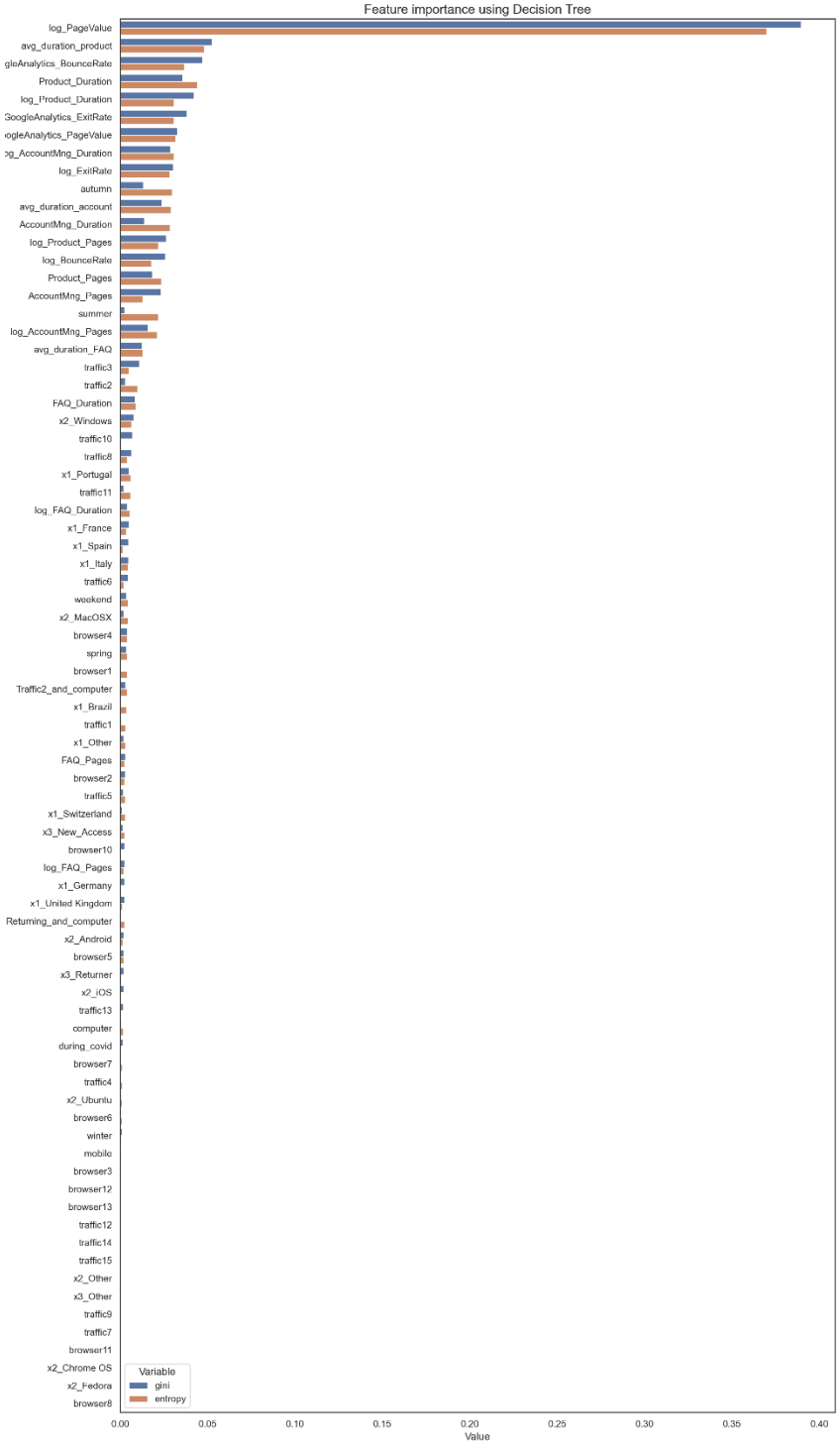

Figure 11: Decision tree for feature selection

- Modeling Visualizations

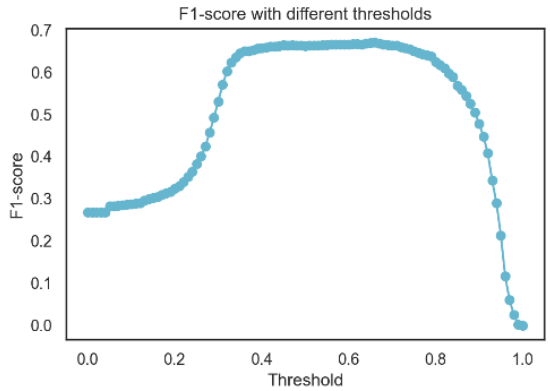

Figure 12: Logistic regression: threshold with outliers

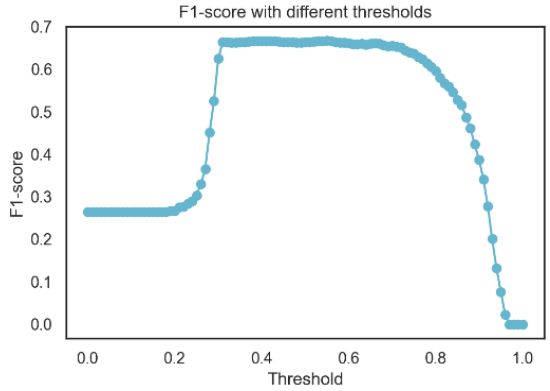

Figure 13: Logistic regression: threshold without outliers

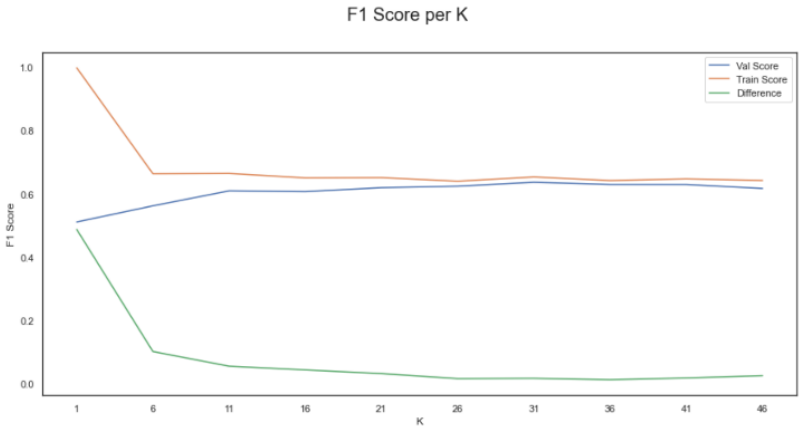

Figure 14: KNN – f1score per K

- Performance Assessment

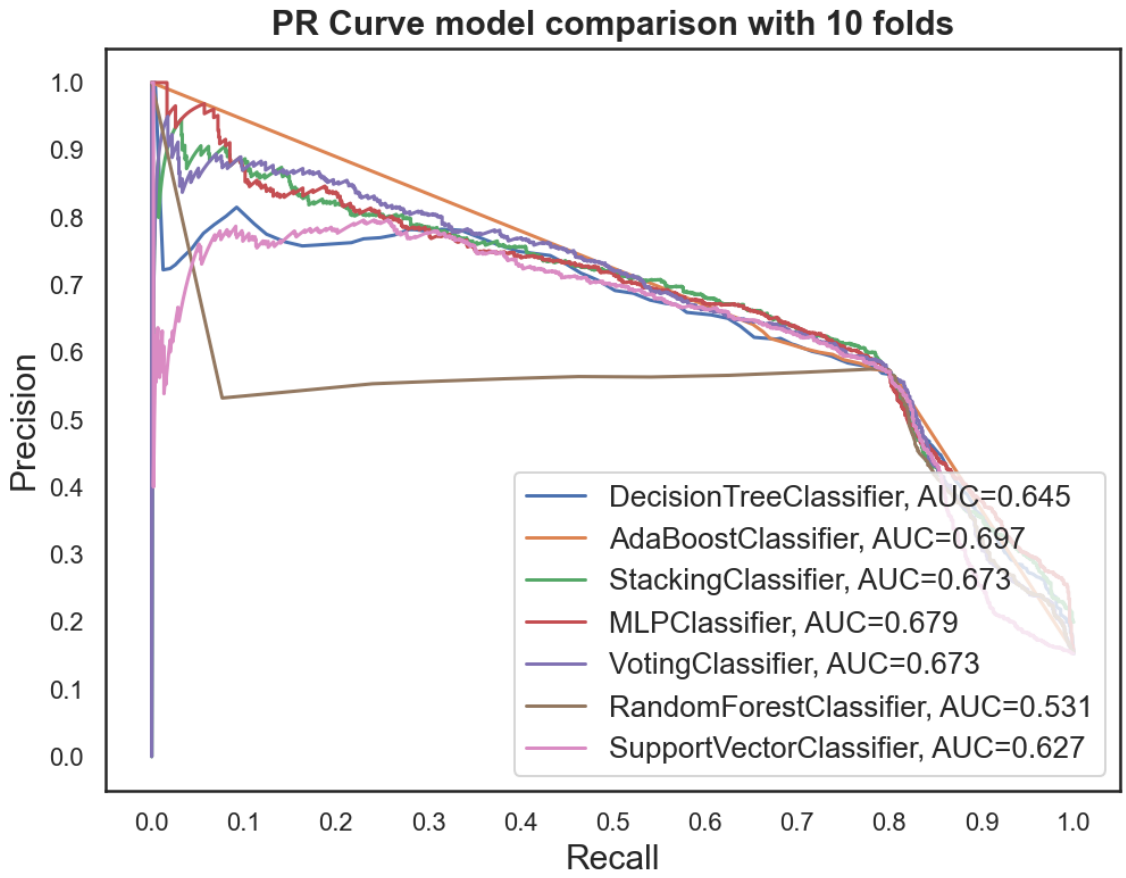

Figure 15: Precision-Recall curve from stratified 10-fold for each model.

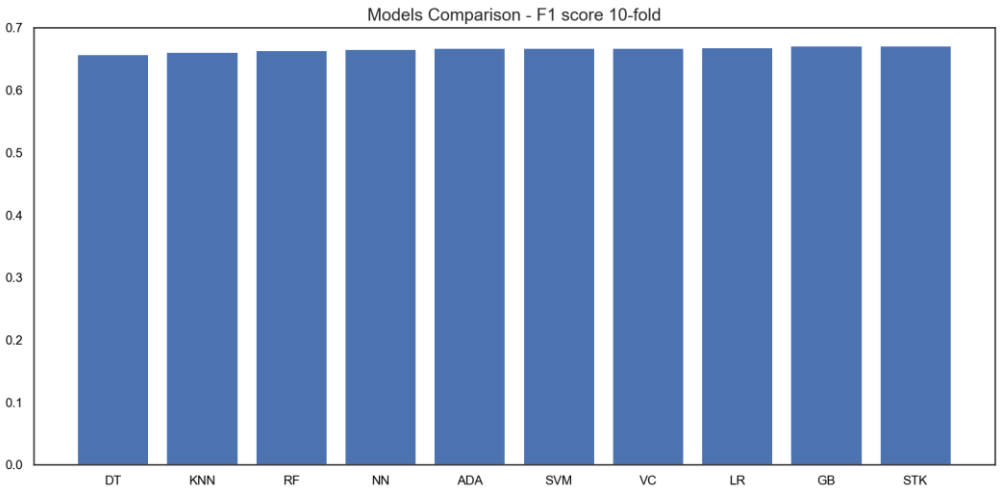

Figure 16: Barplot with F1 score from stratified 10-fold for each model (ordered).

Tables

- Coherence Check

Coherence Number	Rule	Explanation
1	groupby(date.dt.month).agg(min,max)[day]	Check if the maximum and minimum values of day are between 1 and 30 or 31, according to the month
2	(i_duration > 0) & (type_i ==0) where i_duration = {FAQ_Duration, AccountMng_Duration, Product_duration} type_i_pages = {AccountMng_Pages, FAQ_Pages, Product_Pages}	If the consumer did not visit a page of the type *i*, the duration he spent in those pages cannot be greater than 0
3	(Type_i > 0) & (i_duration == 0) where i_duration = {AccountMng_Duration, FAQ_Duration, Product_duration} type_i = {AccountMng_Pages, FAQ_Pages, Product_Pages}	If the consumer did not spend any time in pages of the type *i*, the number of that pages visited cannot be greater than 0
4	*0 < GoogleAnalytics_ExitRate < 1* *0 < GoogleAnalytics_BounceRate < 1*	Rates should be values between 0 and 1
5	Date < dt.datetime(2020,12,31) Date > dt.datetime(2020,1,31)	Our dataset only has records between February and December 2020

Table 1: Coherence Check

- Feature Engineering

Name	Explanation
Month	Month in which the visit to the website occurred
Summer, Autumn, Winter, Spring	Binary variable, equal to 1 if the season is summer, autumn, winter, or spring, respectively
Mobile	Binary variable, equal to 1 if the operating system belongs to a mobile device
Computer	Binary variable, equal to 1 if the operating system belongs to a computer device
Traffic2_and_computer	Binary variable, equal to 1 if type of traffic is 2 and the device used is computer
Returning_and_computer	Binary variable, equal to 1 if the customer is a returner and the device used is a computer
Weekday	Returns the day of the week. 0 is Monday, 1 is Tuesday, …
Weekend	Binary variable, equal to 1 if weekend
avg_duration_account	Returns the average duration of visits to Account Management pages
avg_duration_FAQ	Returns the average duration of visits to FAQ pages
avg_duration_product	Returns the average duration of visits to Product pages

log_AccountMng_Duration, log_AccountMng_Pages, log_FAQ_Duration, log_FAQ_Pages, log_Product_Duration, log_Product_Pages, log_BounceRate, log_ExitRate, log_PageValue	Logarithm to the base 10 of the variables *AccountMng_Duration, log_AccountMng_Pages, log_FAQ_Duration, log_FAQ_Pages, log_Product_Duration, log_Product_Pages, log_BounceRate, log_ExitRate, log_PageValue* [1]
during_covid	Binary variable, equal to 1 if the visit to the website was after the pandemic start

Table 2: Feature Engineering

- Feature Selection

set1	set2	set3	large_set_new	set_new1	set_new2
'AccountMng_Pages'	'AccountMng_Pages'	'AccountMng_Pages'	'log_PageValue'	'AccountMng_Pages'	'AccountMng_Pages'
'Product_Duration'	'Product_Duration'	'Product_Duration'	'log_Product_Duration'	'Product_Duration'	'Product_Duration'
'log_PageValue'	'log_PageValue'	'log_PageValue'	'log_AccountMng_Duration'	'log_PageValue'	'log_PageValue'
'x3_New_Access'	'x3_New_Access'	'x2_MacOSX'	'log_BounceRate'	'GoogleAnalytics_ExitRate'	'GoogleAnalytics_ExitRate'
'traffic2'	'traffic2'	'x2_Windows',	'log_ExitRate'	'log_FAQ_Pages'	'log_FAQ_Pages'
'during_covid'	'during_covid'	'during_covid'	'avg_duration_FAQ'	'summer'	'Traffic2_and_computer'
	'GoogleAnalytics_ExitRate',	'GoogleAnalytics_ExitRate',	'autumn' 'summer' 'weekend' 'computer'	'Returning_and_computer'	'Returning_and_computer'
	'log_FAQ_Pages'	'log_FAQ_Pages'		'weekend'	'winter'

Table 3: Sets of variables to use on modelling.

Note: another set of variables were defined with the metric features from set1 and set2

[1] 1 is added to the variable so that it is avoided infinite values provoked by 0 in the true value of the variable

Binary variables	Chi-Squared test	Variance Threshold
x2_MacOSX	1	1
x2_Windows	1	1
x3_New_Access	1	1
traffic2	1	1
summer	1	1
autumn	1	1
winter	1	1
spring	1	1
Traffic2_and_computer	1	1
Returning_and_computer	1	1
weekend	1	1
during_covid	1	1
x1_Portugal	0	1
x2_Android	0	1
x1_United Kingdom	1	0
x2_iOS	0	0
x3_Other	0	0
x3_Returner	1	0
traffic1	0	1
traffic3	1	0
traffic11	1	0
traffic13	1	0
traffic15	1	0
traffic8	1	0
browser2	0	1
'browser13	1	0
computer	1	0
mobile	1	0

Table 4: Selection of binary variables

Metric Features	RFE	Forward	Ridge	DT
'log_PageValue'	1	1	1	1
'log_Product_Duration'	1	1	0	1
AccountMng_Pages'	1	1	0	0
'Product_Duration'	0	1	1	0
'GoogleAnalytics_BounceRate'	1	0	0	1
'GoogleAnalytics_ExitRate'	1	1	0	0
'log_AccountMng_Duration'	1	1	0	0
'log_AccountMng_Pages'	1	1	0	0
'log_FAQ_Pages'	1	0	1	0
'log_Product_Pages'	1	0	0	1
'avg_duration_FAQ'	0	0	1	0
'GoogleAnalytics_PageValue'	1	0	0	1
'avg_duration_account'	0	0	0	1

'avg_duration_product'	0	0	0	1
'log_FAQ_Duration'	1	0	0	0
'log_BounceRate	1	0	0	0
'log_ExitRate'	1	0	1	0
'AccountMng_Duration'	0	0	0	0
'FAQ_Pages'	1	0	1	0
'FAQ_Duration'	0	0	0	0
'Product_Pages'	1	0	0	0

Table 5: Selection of metric variables

	No Outliers Dataset			Outliers Dataset		
Model	**F1 score – Stratified 10-Fold**	**F1 score Train**	**F1 score Validation**	**F1 score – Stratified 10-Fold**	**F1 score Train**	**F1 score Validation**
Decision Tree	0.656	0.664	0.654	0.644	0.664	0.650
Neural Network	0.658	0.650	0.642	0.665	0.666	0.656
Logistic Regression	0.667 [2]	0.663	0.655	0.669 [2]	0.670	0.650
KNN	0.660 [2]	0.672	0.658	0.624 [2]	0.681	0.601
Support Vector Machine	0.667	0.667	0.653	0.665	0.665	0.652
Random Forest	0.662	0.664	0.655	0.665	0.664	0.653
Adaboost	0.665	0.667	0.652	0.667	0.668	0.652
Gradient Boost	0.626 [2]	0.648	0.661	0.670 [2]	0.646	0.663
Voting	----	----	----	0.667	0.669	0.652
Stacking Classifier	----	----	----	0.671	0.668	0.643

Table 6: Models comparison

[2] These values represent the F1-score using a different function, which uses the predictions using the best threshold to compute the F1-score. Even though it is known that using cross-validation various datasets are considered, which can imply various optimum thresholds, it was decided to use this function to compare models, as it was verified that the values didn't change much.

Model	Variable Set	Dataset	Best F1 score
Decision Tree	Set3	No outliers	0.656
Neural Network	Large_set_new	Outliers	0.665
Logistic Regression	Set_new2	Outliers	0.669
KNN	Set1_no_bin	No outliers	0.660
Support Vector Machine	Set1_no_bin	No outliers	0.667
Random Forest	Sets_RF1	Outliers	0.665
Adaboost	Set_new1	Outliers	0.667
Gradient Boost	Set_new1	Outliers	0.670
Voting	Set_new1	Outliers	0.667
Stacking Classifier	Large_set_new	Outliers	0.671

Table 7: Datasets and variable sets used in models

Train Dataset				
	Precision	**Recall**	**F1-Score**	**Support**
0	0.93	0.95	0.94	5913
1	0.68	0.64	0.66	1086
Accuracy			0.90	6999
Macro Avg	0.81	0.79	0.80	6999
Weighted Avg	0.90	0.90	0.90	6999
Validation Dataset				
	Precision	**Recall**	**F1-Score**	**Support**
0	0.93	0.94	0.93	2534
1	0.65	0.64	0.64	466
Accuracy			0.89	3000
Macro Avg	0.79	0.79	0.79	3000
Weighted Avg	0.89	0.89	0.89	3000

Table 8: Stacking model metrics