NOVA IMS

Machine Learning

Report

Group 29

Ana Luís | 20210671

Carolina Machado | 20210676

Francisco Calha | 20210673

Luís Santos | 20210694

Sara Arana | 20210672

TechScape

Which customers are more likely to buy?

Fall 2021

INDEX

1. Abstract

Our goal with this project is to increase the sales for the company TechScape, by identifying the 'Buyers' from 'Non-Buyers' from the data regarding the online behavior of possible customers. We aimed at achieving the best possible estimator for our model, to increase the accuracy of our predictions in the Test dataset. Before selecting and assessing the models, we started by exploring the data pre-processing process. We went through feature engineering, outlier detection methods, encoding of categorical variables, and feature selection. To improve our results, we tried over and under sampling techniques and different scalers. After, we explored different models, such as Decision Tree, Logistic Regression, Bagging, Gradient Boosting, among others. Our best model ended up being the Histogram-based Gradient Boosting, and we obtained a f1 score of 0.7862 and 0.6610 on the training and validation datasets, respectively.

2. Introduction

TechScape is a startup company founded in the beginning of 2020, by five Portuguese entrepreneurs, with the goal of selling products related to digital detox. Due to the fast-paced technological development era we are living in, it is required to be able to disconnect; as such, TechScape fits the market necessities. However, during the first lockdown the company went through financial difficulties and ended up out of business. In May 2020, they restarted their activities. Amidst the pandemic scenario, which started in March 2020, people started spending more time online, and less time outdoors; this intensified the need for the digital detox that TechScape aims to achieve.

Our target is to understand which customers will purchase the products sold in TechScape, by building a predictive model, using the data available from the customers database (from February 2020 until December 2020, excluding April 2020). With this project, we tried to focus on having the most realistic prediction, rather than focusing on the score we were obtaining in Kaggle. We looked for reducing overfitting as much as we could.

3. Methodology

3.1. Datasets and Variables' Description

We were provided two datasets, a training dataset, and a testing dataset. The first one had the target variable 'Buy', which is a binary variable with values 0 or 1 (did not buy or buy, respectively). This target variables aims to predict is a customer made a purchase or not, so, we are dealing with a classification problem. Our initial training dataset had 9999 observations and 16 variables (Table 1 for variables description and Table 2 for variable types). Each observation is identified by the Access ID, which is the unique identifier of the user access to the website.

3.2. Imports

We imported the required libraries for our project, namely Pandas, NumPy, Matplotlib, Seaborn, Math, SciPy, and Sklearn. Afterwards, for the pre-processing part, we imported pre-processing libraries (MinMaxScaler, StandardScaler, RobustScaler, and OneHotEncoder). For feature and model selection, we imported several libraries from Sklearn (e.g., RFECV, LassoCV, DecisionTreeClassifier, MLPClassifier, etc.)

3.3. Data Exploration

When beginning the data exploration step, we analyzed our response variable ‘Buy’, and we understood that we are dealing with an unbalanced dataset, since 84,48% of our observations are 0’s and only 15,52% are 1’s. This implies that the buyers of our dataset are the minority class. This problem will be addressed later on the report.

3.3.1. Missing Values

To start exploring the data, we used methods such as describe and info. We concluded that we had no missing values in any of the columns of our dataset, however, the feature’s types were not correct in some cases (i.e., we defined ‘Type_of_Traffic’ and ‘Browser’ as ‘objects’).

3.3.2. Inconsistencies

Then, we checked for possible inconsistencies. Firstly, we checked whether the ‘Date’ variable was within the pre-defined range (between February and December of 2020, except for the month of April), and everything was correct. Secondly, we checked whether there were observations in which the number of pages visited was 0 and the amount of time spent on those pages was different than 0, we also concluded everything was consistent (we checked this for ‘AccountMng_Pages’ and ‘AccountMng_Duration’, ‘FAQ_Pages’ and ‘FAQ_Duration’, and ‘Product_Pages’ and ‘Product_Duration’).

3.3.3. Defining Independent Variables and Target

Afterwards, we proceeded to define our independent and target variables into two different objects (‘independent_variables’ and ‘target’, respectively).

3.3.4. Descriptive Statistics

We used the Describe function to explore statistics for our metric features (Table 3). From this, we concluded that there were outliers in some of the variables (‘AccountMng_Duration’, ‘FAQ_Pages’, ‘FAQ_Duration’, ‘Product_Pages’, ‘Product_Duration’, ‘GoogleAnalytics_PageValue’). For example, in ‘AccountMng_Duration’, we can see that the 50% quantile value and the maximum are very sparse, hence, there are clearly outliers that need to be possibly removed from this variable. The same reasoning applies to the other mentioned features. In addition, by comparing the mean and the median from this table, we can also expect our variables to follow a skewed distribution.

For the categorical features, we used the same function. However, to understand them better we had to perform some feature engineering on some variables (see **3.4.1. Feature Engineering**).

3.4. Data Preparation

3.4.1. Feature Engineering

Regarding feature engineering, we started by creating a new variable 'Month' which corresponds to the month of each date, from the previous 'Date' column. Afterwards, we concluded that the 'OS' variable would lead to better and more simpler conclusions if we group the operating systems of computers, cellular, and others. The operating systems of the computers are MacOSX, Windows, Ubuntu, Fedora, Chrome OS; and the operating systems of the cellular are Android, and iOS. The 'Other' observation in the 'OS' variable was replaced by the mode - iOS. We decided to replace these values by the mode because they represented a very small percentage of the 'OS' column. For the variables 'Month', 'Country', 'Browser', and 'Type_of_Traffic', we noticed that there were some observations with low cardinality, so we decided to join those observations – 'Other Date', 'Other Country', 'Other Browser', and 'Other Type' respectively. We grouped these minorities because since we are going to split our dataset into two different datasets (train and validation) we might end up with one of them with certain observations and the other with none of them. Regarding this idea of combining certain observations in one label, we first tried a softer combination, by only joining those whose cardinality was close to 0 (using the countplot function). After, we tried a more restrict rule, by increasing the threshold. The latter, led to a reduction in the input space, and, consequently, to better results.

From table 4, we can see the influence that the most frequent observations have in each of the non-metric features. For instance, in the 'OS' variables, the most frequent observation is 75.2%, meaning more than half of the individuals in our dataset use their PC to go on TechScapes's website. On the other hand, regarding the 'Month' variables, May is the month with most website visits, however its percentage is only 27.3%. Annex 1 helps us visualize better the distribution of our categorical variables.

3.4.2. Data Partition

To build our predictive model, it is necessary that we split out dataset into two different datasets. We used the train_test_split function, from which 80% of the observations were in the train dataset and 20% were in the validation dataset – 7999 observations for training and 2000 observations for validation. We defined the stratify parameter equal to the target, for the train and validation data sets to preserve the same proportion of examples in each class, as it is observed in the original dataset. Random State was set to 1, to fit and evaluated the same subsets of the dataset, each time we run the models. Finally, the parameter Shuffle was also set to True, to avoid bias in the datasets, by introducing some randomness.

3.4.3. Outliers

We experimented our model with and without outliers. Our goal is never to surpass the 3% threshold of outliers' removal.

To detect the outliers in metric features, we plotted boxplots and histograms for all these variables (Annex 2 and 3). From the histograms, we verified our initial assumption about the skewed distribution of our numerical variables.

We started by trying the IQR method to remove outliers, but since some variables had a significant number of values at the extremes, we were eliminating too many observations. So, we figured the best approach would be to establish, manually, thresholds for the most ‘extreme’ outliers, based on the visualizations. From the plots, we could not see very clearly the right threshold. So, for some of them, we plotted the boxplot excluding 0, to be more accurate. Being able to better analyze the observations, we started by removing the extreme observations from our dataset.

Since we were not able to eliminate all global outliers with the previous method, we decided to use the DBSCAN method. We were able to focus also on possibly forgotten global outliers. To apply this algorithm to our model, we defined the number of neighbors to be 100, and we scaled a copy of our data using the MinMax Scaler – because we needed a scaler that was sensitive to outliers. Since we have a very small number of 1’s in the target variable, we applied the DBSCAN and only eliminated the outliers for which the response if 0. After this, we were able to remain with around 97.58% of the observations.

Despite having very similar results, we ended up using the dataset with outliers because it led to a better overall performance of our model.

3.4.4. Encoding Categorical Variables

To build a better predictive model, including our categorical variables, to do so we used the OneHotEnconder method. This method converts each different label, for each of the categorical variables, into a binary vector – i.e., each new columns takes the values of 0 or 1. From table 5, we can see the newly created features, after the variable encoding process. This process was applied to the validation and test datasets.

3.4.5. Feature Selection

On a first instance, we computed the Spearman Correlation matrix (Annex 4), to check whether there were any redundancies in our features. Since there were 4 pairs of highly correlated variables (threshold of 80%) and decided to eliminate one in each pair. To decide on which variables to drop we first looked at the Random Forest feature importance. But, we ended up deciding based on the correlation of the pairs of highly correlated features with the remaining variables – this led to better results. We eliminated the following features: 'AccountMng_Duration', 'FAQ_Pages', 'Product_Duration', 'GoogleAnalytics_ExitRate'.

Regarding these highly correlated variables, we thought about the hypothesis of joining them, because by doing so, we would be reducing the input space and we would not lose their information. Our idea was to divide the variables that are related to time spent on each page by the number of pages search. From here, we would have the average time per page for ‘AccountMng’, ‘Product’, and ‘FAQ’ pages. We decided that the 0 values in both variables and 0 in the number of pages would both be set to 0, in our new variables. The results using these features were not favorable in our model, so we did not proceed with this idea.

Afterwards, we started to explore a few feature selection methods.

Random Forest

We decided to use the Random Forest feature selection method on our model. This method has two splitting criterions, the Gini Index (impurity) and Entropy (information gain), which are useful to select the most relevant features. It calculates those values at each step, and, for each node of the tree, it creates a subset of the most relevant features. We defined a threshold of 0.055, meaning all features below this value would not be optimal for our model (see Annex 5). The subset of features we achieved here were 'AccountMng_Pages','GoogleAnalytics_BounceRate', 'GoogleAnalytics_PageValue', 'Product_Pages'. Notice that the values did not differ significantly from Gini to Entropy, so we kept with the default value (Gini).

Lasso Regression

On a second instance, we performed the Lasso regression. Its objective is to achieve a set of features that minimize the prediction error for a quantitative target variable. To obtain this subset of variables, this method imposes a constraint on the model parameters that causes the coefficients for some features to shrink to zero. From the Lasso regression we got the following features 'AccountMng_Pages', 'FAQ_Duration', 'GoogleAnalytics_BounceRate', 'GoogleAnalytics_PageValue', 'Product_Pages', 'May', 'November', 'Other_Month', 'Returner', 'Traffic_2', 'Traffic_3' (see Annex 6).

A-NOVA, Chi-Squared and Mutual Information (Combination Features)

In order to use the A-NOVA feature selection method, we decided to combine it with the Chi-Squared and Mutual Information methods – since the last two methods are for categorical input and output variables, and the A-NOVA, in our case, works for numerical input variables.

To perform the A-NOVA test, we computed the test statistic values for all metric features, and we saw that there was one variable that was significantly more relevant than the others, ‘GoogleAnalytics_PageValue’ (see Annex 7).

For categorical variables we first performed the Chi-squared test, for which the null hypothesis is that the observed frequencies for a categorical variable match the expected frequencies for the categorical variable. The model automatically returns which features the model should keep or discard. Secondly, we performed the Mutual Information method, in which it is measured the reduction in uncertainty for one variable, given a value of the other variable. For this approach, we defined a threshold of 0.003, for which we kept the features whose value was above the threshold (see Annex 8).

When combining all these methods, we got the following features 'Traffic_5', 'Traffic_2', 'March', 'Traffic_10', 'Traffic_15', 'November', 'Traffic_13', 'Traffic_11', 'Returner', 'Traffic_3', 'Traffic_8', 'May', and 'GoogleAnalytics_PageValue'.

Recursive Feature Elimination CV

To finalize the feature selection process, we performed the Recursive Feature Elimination method with Cross Validation. To obtain the best possible set of features, we explored the most common algorithms to help choose the features, namely Logistic Regression, Decision Tree Classifier, Gradient Boosting Classifier, and Random Forest Classifier. For each algorithm, we used a Stratified K Fold process, with 5 splits, and we obtained the optimal number of features for each of them. We opted for the Stratified K Fold since we have an imbalanced dataset, and the stratified characteristic ensures that each fold has equal proportions of observations of 0's and 1's

The features selected for each model, using the RFE method are represented in table 5. We decided to use the RFE feature selection method only one algorithm – to reduce computational costs -, Gradient Boosting, since it had the best score. The chosen features from this method were the following: 'AccountMng_Pages', 'FAQ_Duration', 'GoogleAnalytics_BounceRate', 'GoogleAnalytics_PageValue', 'Product_Pages', 'November', and 'Returner'.

AdaBoost Feature Importance

Finally, we used the AdaBoost Feature Importance technique, which works similarly to the Random Forest method, since it follows the reasoning that the features used at the top of the tree are more important. The only difference between the methods is the base classifier. From here, we got the following features (Annex 9): 'AccountMng_Pages', 'FAQ_Duration', 'GoogleAnalytics_BounceRate', 'GoogleAnalytics_PageValue', and 'Product_Pages'.

Other Failed Feature Selection Methods Explored

Before using the Random Forest feature importance method to define our variables, we started by using the Decision Tree Classifier for the same purpose. But we ended up using Random Forest since it is more accurate, and an improved version of the Decision Tree.

Another method explored for feature selection was the Ridge Regression, which was applied to all features in the dataset. The Ridge Regression coefficients are different from Lasso coefficients since they do not shrink to 0, but to values very close to 0. To select the features using the ridge coefficient, we defined a threshold that made sense in our data; and the features selected were the ones whose coefficients were outside this range. Notice that, for our data, the ridge regression did not lead to very good results. This could be related to the fact that only a few sets of features are relevant for our problem.

We also tried to apply several methods and try to define the features based on a voting regarding what most of the methods stated about the importance of those features, as we did on the practical classes. We did not follow through with this idea, because the results obtained were not optimal.

3.4.6. Normalization

For the normalization part of our pre-processing step, we considered using 4 different scalers: StandardScaler, MinMaxScaler(0,1), MinMaxScaler(-1,1), and RobustScaler. Using different scaling methods led to very different results; so, by following a trial-and-error process, we ended up using the RobustScaler methods, from which the results obtained were more favorable to our problem. This method uses the interquartile range so that it is robust to outliers. Its formula is as follows:

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

3.4.7. Under sampling and Oversampling

On a final step for data preparation, we concluded it was required for us to apply an under sampling or oversampling technique. By looking at our recall measure, we realized our model was not being able to correctly predict the 'Buyers' (1's). This happened because classification problems, such as the one we are trying to model, often have little observations of the minority class (1's). However, this class is the most important for us since our final goal is to end up increasing sales for the company. Hence, it makes sense to be able to understand where the buyers are in our target customers. There are several under and over sampling techniques to overcome this issue, but we ended up choosing the Synthetic Minority Oversampling Technique (SMOTE) approach. This method differs from the others because it generates new observations based on the Euclidean distance of each data and the minority class nearest neighbors, so, the new examples are different observations from the ones originally from the minority class. Besides SMOTE, we tried Tomek Links method for under sampling, and combining SMOTE with Tomek Links. But our best results were when we used the SMOTE approach.

To avoid further overfitting, we applied SMOTE inside the Cross Validation process.

4. Model Selection and Assessment

After the feature selection process, we began exploring the different possible models for our predictions. We started by calculating the average f1 score for Stratified K Fold cross validation (with 5 splits) for several classifiers – Logistic Regression, Decision Tree, Neural Networks, Gaussian Naïve Bayes, and K Nearest Neighbors. The metric used is the weighted average of Precision and Recall, and it is helpful in cases of uneven class distribution, as ours.

$$\text{F1 Score} = \frac{2 \text{ x (Precision x Recall)}}{\text{Precision + Recall}} \qquad \text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

From table 6, we can see the f1 scores for each of these models, and we can see that most of them are overfitted, as expected since we are using SMOTE. The ones in bold are those for which we will perform hyper-parameter tunning, to obtain the best set of parameters and try to reduce the overfitting. The chosen models for hyper-parameter tunning were the Decision Tree Classifier and Neural Networks. These models are those that we already expected to have overfitting, because it is how they usually behave. For this part, we opted to use HalvingGridSearchCV rather than GridSearchCV because it still finds the best parameters but in less time, so we were able to reduce computational costs.

From table 7, we can see the new cross validation scores for the Decision Tree Classifier and Neural Networks.

Next, we decided to choose the 3 classifiers with highest validation f1 score, for each set of features. With the top 3 weak learners found previously, we computed the f1 average score for the ensemble classifiers. In table 8, we can see the cross-validation results for the following models: Voting Classifier, Stacking Classifier, AdaBoost Classifier, Bagging Classifier, Random Forest, Gradient Boosting, and Histogram-based Gradient Boosting. For the Bagging Classifier, we only used the best weak classifier which was the Neural Network. We even computed the hyper-parameter tunning for Gradient Boosting, Histogram-Based Gradient Boosting, and Random Forest (table 9).

We decided to explore the Histogram-based Gradient Boosting classifier because it is an improved and faster version of the Gradient Boosting classifier. By putting continuous feature values into discrete bins, the model becomes faster and might even improve the results (as it happened to us).

From table 10, we can see the best model for each set of features, and here we can see the actual validation and training f1 scores and the recall measure for a the best cutoff value.

5. Conclusions

To conclude, our best model, at least in terms of overfitting and prediction of the minority class (1's), is the Histogram-based Gradient Boosting, using the Lasso Regression features. We set the following parameter: learning_rate at 0.03, max_depth at 3, max_leaf_nodes at 12, random_state at 1, and scoring as 'f1'. In addition, we tried to find the optimal cutoff value for the probability of belonging to class '1', and we changed the default value (0.5) to 0.557, since it was the value for which we got a better validation score.

Using this model, we obtained a 0.68518 score in Kaggle (since have access to our Test dataset, we were able to see that it was the best Kaggle score obtained from those in Table 9). Although not being our best obtained score, it was the one we believed to be more reasonable. The higher scores we got previously were related to the fact that our model was overfitted, and we tried to overcome this issue, by using the average f1 score from the cross-validation function, with the SMOTE incorporated. From here, we expect that the results from the public score will be similar to the ones in the private score. Our model has a recall of 75.48, which means that it matched 75% of the 1's in the validation dataset.

There are several steps we could have done to increase the performance of our model. Namely, conduct a more detailed hyper parameter tunning for the Bagging and Gradient Boosting Classifiers and by using the Support Vector Machine Classifier, however, these operations were very time consuming, so we decided not to follow through.

6. Annexes

Table 1 – Description of Variables

Attribute	Description
Date	Website visit date.
AccountMng Pages	Number of pages visited by the user about account management.
AccountMng Duration	Total amount of time (seconds) spent by the user on account management related pages.
FAQ Pages	Number of pages visited by the user about frequently asked questions, shipping information and company related pages.
FAQ Duration	Total amount of time (seconds) spent by the user on FAQ pages.
Product Pages	Number of pages visited by the user about products and services offered by the company.
Product Duration	Total amount in time (seconds) spent by the user on products and services related pages.
GoogleAnalytics BounceRate	Average bounce rate value of the pages visited by the user, provided by google analytics.
GoogleAnalytics ExitRate	Average exit rate value of the pages visited by the user, provided by google analytics.
GoogleAnalytics PageValue	Average page value of the pages visited by the user, provided by google analytics.
OS	Operating System of the user.
Browser	Browser used to access the webpage.
Country	The country of the user.
Type of Traffic	Traffic Source by which the user has accessed the website (e.g., email, banner, direct).
Type of Visitor	User type as “New access”, “Returner” or “Other”.
Buy	Class label indicating if the user finalized their actions in the website with a transaction.

Table 2 – Types of Variables

Categorical Variables	**Nominal Variables**	Date
		Country
		OS
		Type_of_Traffic
		Browser
		Type_of_Visitor
Numerical Variables	**Discrete Variables**	AccountMng_Pages
		FAQ_Pages
		Product_Pages
	Continuous Variables	AccountMng_Duration
		FAQ_Duration
		Product_Duration
		GoogleAnalytics_BounceRate
		GoogleAnalytics_ExitRate
		GoogleAnalytics_PageValue

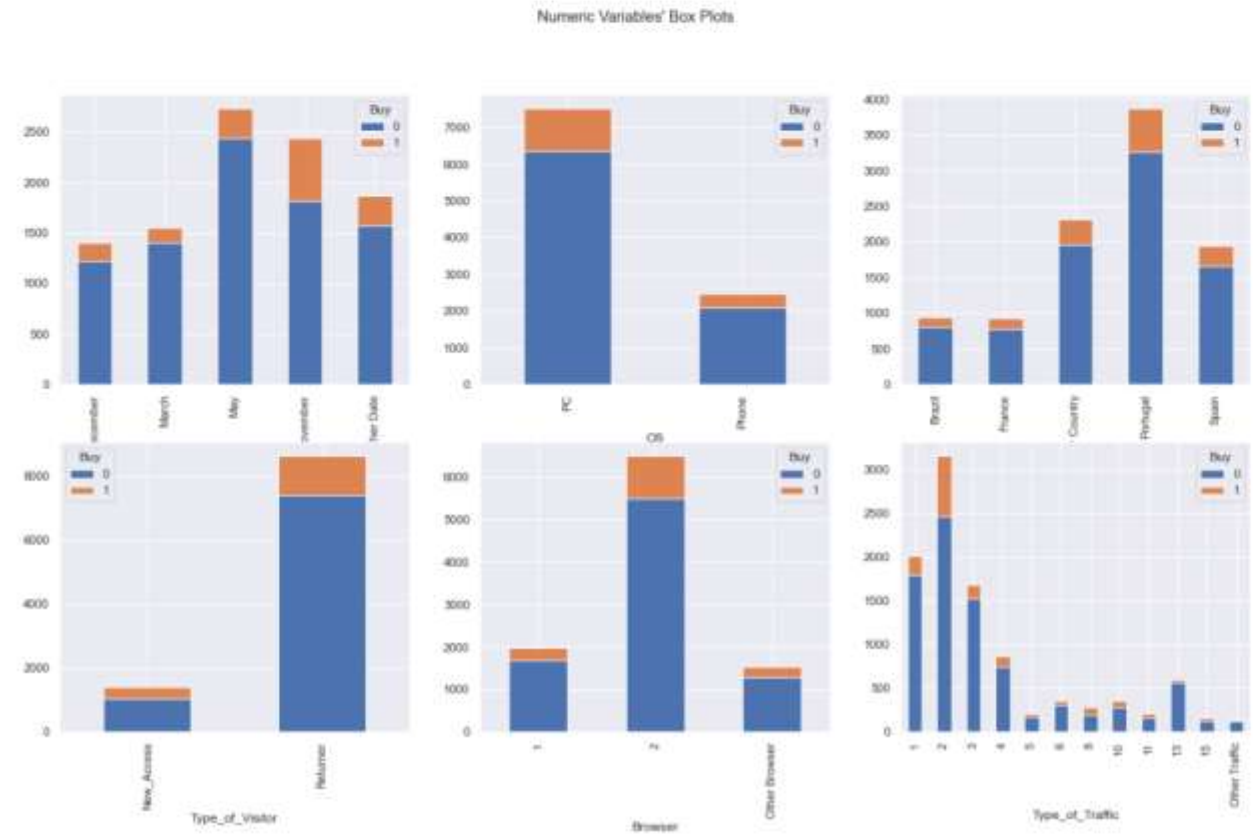
Table 3 – Descriptive Statistics (numerical features)

	count	mean	std	min	25%	50%	75%	max
AccountMng_Duration	9999	81,20585434	179,715545	0	0	7,5	92,20835	3398,75
AccountMng_Pages	9999	2,324232423	3,34067596	0	0	1	4	27
FAQ_Duration	9999	34,55910121	139,796989	0	0	0	0	2549,375
FAQ_Pages	9999	0,508050805	1,279389526	0	0	0	0	24
GoogleAnalytics_BounceRate	9999	0,022305451	0,048775974	0	0	0,0032	0,0168	0,2
GoogleAnalytics_ExitRate	9999	0,043181468	0,048845276	0	0,0143	0,0251	0,05	0,2
GoogleAnalytics_PageValue	9999	5,963120292	18,75362571	0	0	0	0	361,7637
Product_Duration	9999	1199,76943	1958,276304	0	183,5625	599	1470,2708	63973,5222
Product_Pages	9999	31,68586859	44,55027695	0	7	18	38	705

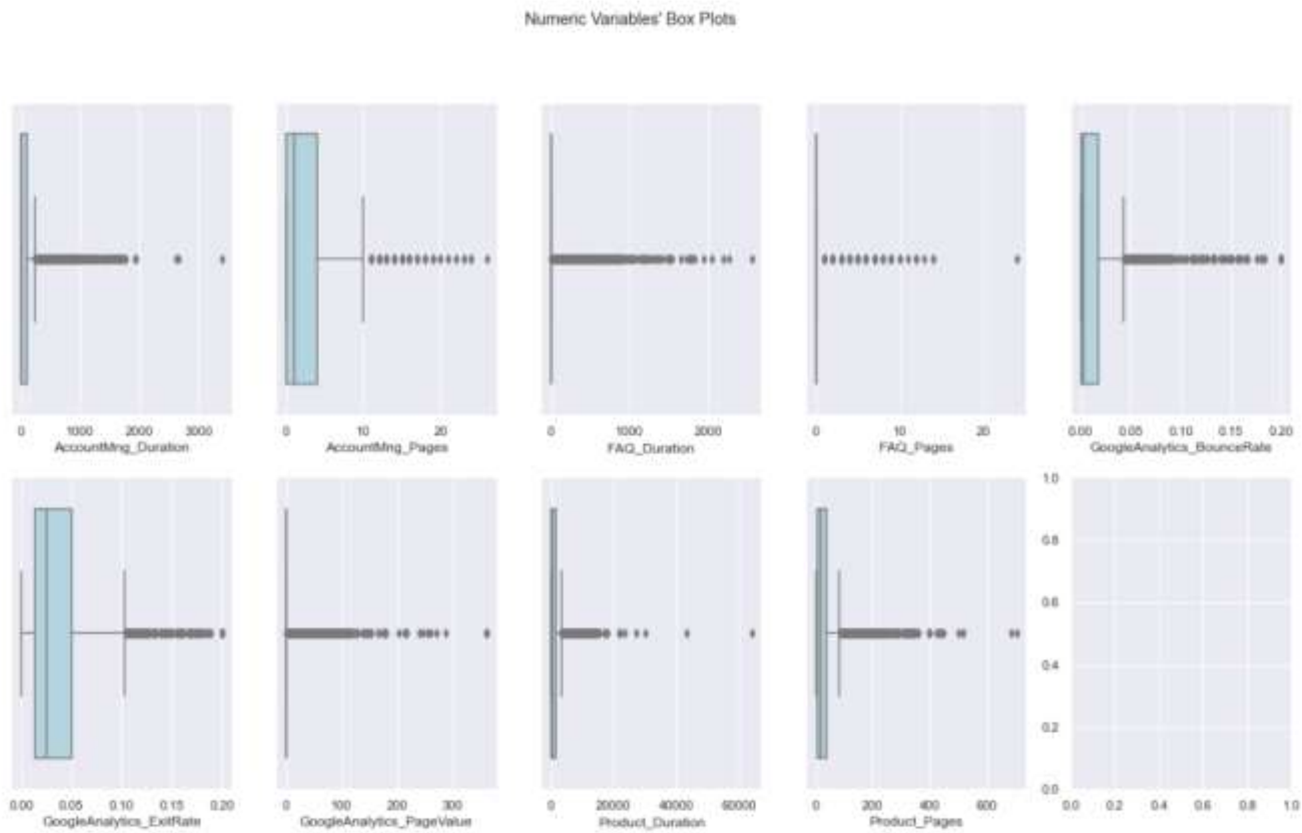
Table 4 – Descriptive Statistics (categorical features)

	Browser	Country	OS	Type_of_Traffic	Type_of_Visitor	Month
count	9999	9999	9999	9999	9999	9999
unique	3	5	2	6	2	5
top	2	Portugal	PC	2	Returner	May
freq	6484	3870	7517	3150	8608	2732
freq of most observed values (%)	64,8%	38,7%	75,2%	31,5%	86,1%	27,3%

Annex 1 – Categorical Variables Frequency Distribution



Annex 2 – Numerical Variables Boxplots



Annex 3 – Numerical Variables Histograms

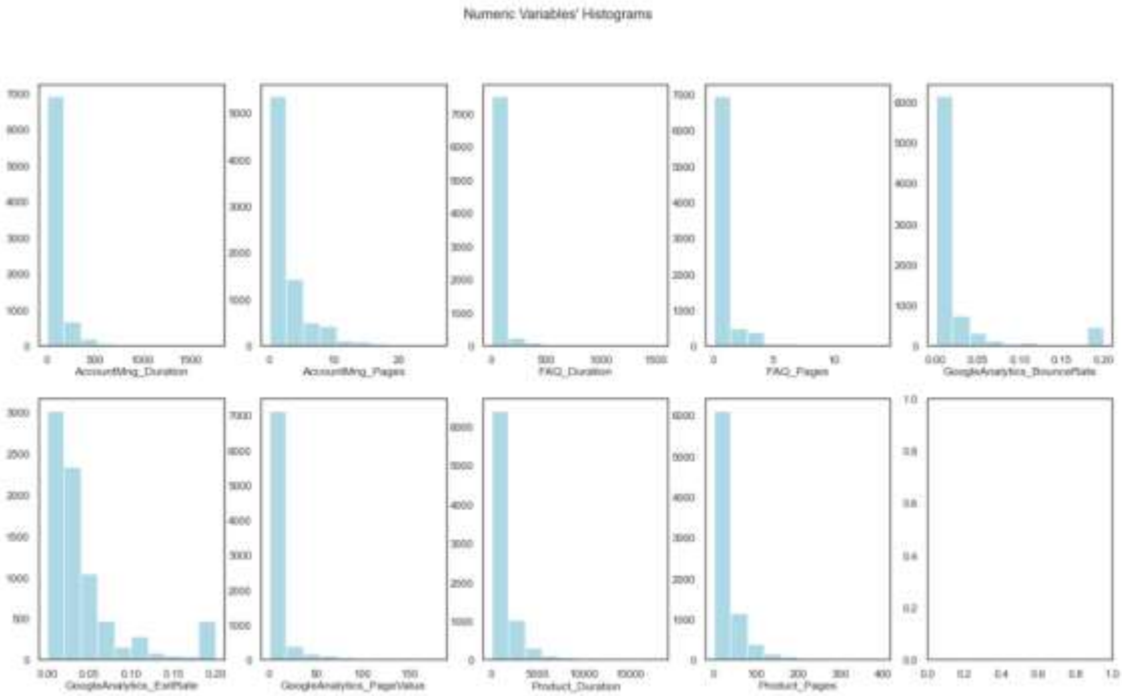
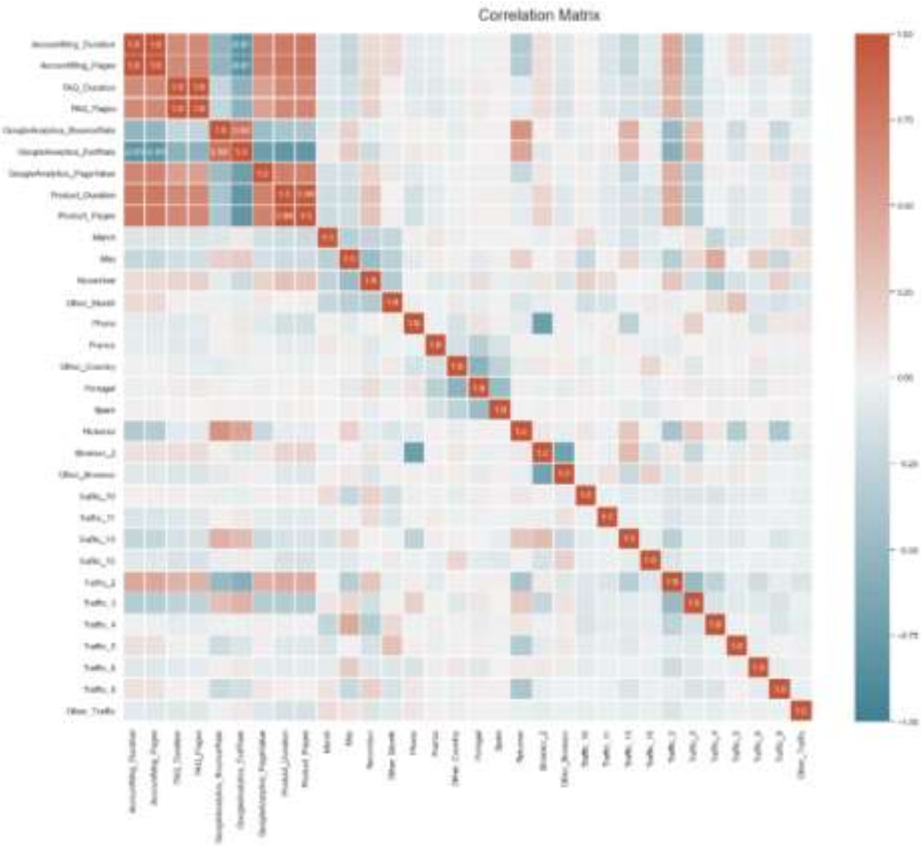


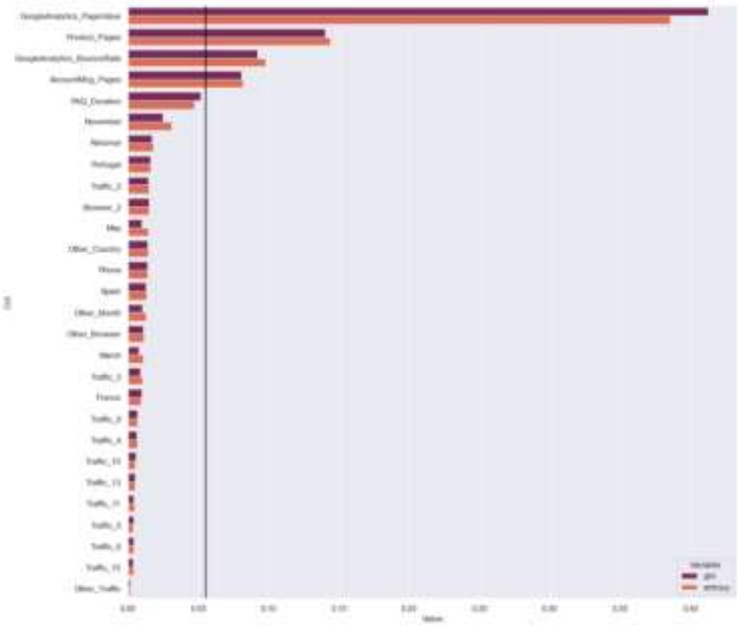
Table 5 – One-hot encoder (training dataset)

Access_ID	March	May	November	Other_Month	Phone	France	Other_Country	Portugal	Spain	Returner	Traffic_13	Traffic_2	Traffic_3	Traffic_4	Other_Traffic	Browser_2	Other_Browser
259448731	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
430925192	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
625441758	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
600270594	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
131813376	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0

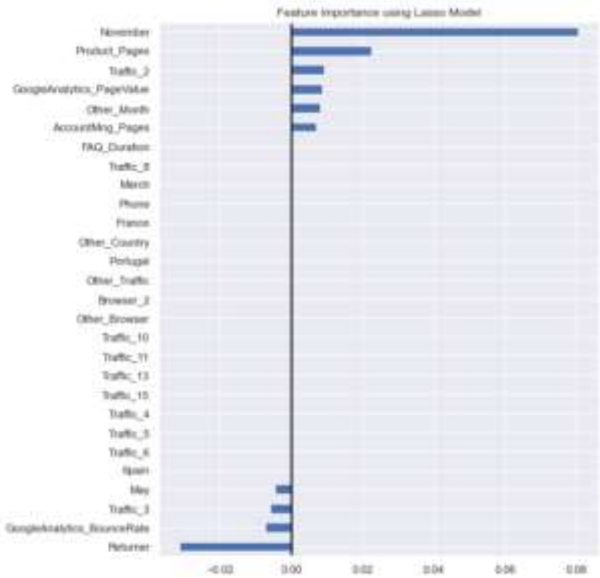
Annex 4 – Spearman Correlation Matrix



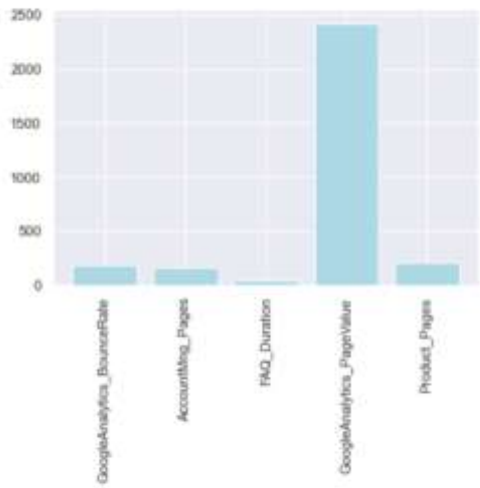
Annex 5 – Random Forest with Gini and Entropy



Annex 6 – Lasso Regression Coefficients



Annex 7 - A-NOVA f-statistic test



Annex 8 – Mutual Information for Categorical Features

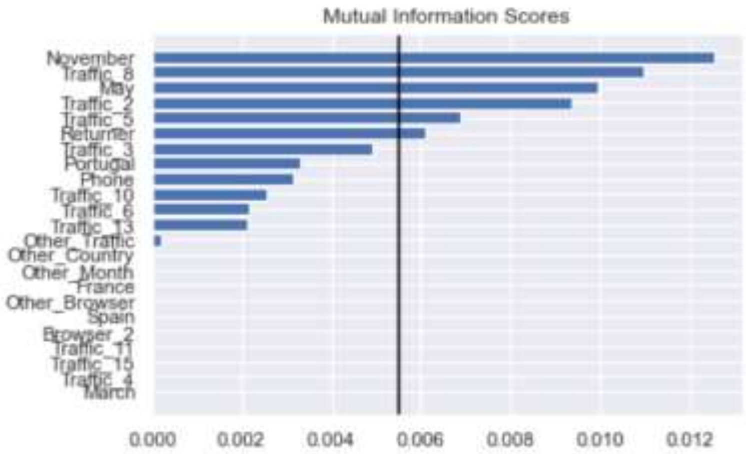


Table 5 – RFE CV important features (for different model)

RFE with Logistic Regression	'AccountMng_Pages', 'FAQ_Duration', 'GoogleAnalytics_BounceRate', 'GoogleAnalytics_PageValue', 'Product_Pages', 'March', 'May', 'November', 'Other_Month', 'Phone', 'France', 'Other_Country', 'Portugal', 'Spain', 'Returner', 'Browser_2', 'Other_Browser', 'Traffic_10', 'Traffic_11', 'Traffic_13', 'Traffic_15', 'Traffic_2', 'Traffic_3', 'Traffic_4', 'Traffic_5', 'Traffic_6', 'Traffic_8', 'Other_Traffic'
RFE with Decision Tree Classifier	'GoogleAnalytics_PageValue', 'Product_Pages'
RFE with Gradient Boosting Classifier	‘AccountMng_Pages', 'FAQ_Duration', 'GoogleAnalytics_BounceRate', 'GoogleAnalytics_PageValue', 'Product_Pages', 'November', 'Returner'
RFE with Random Forest Classifier	'AccountMng_Pages', 'FAQ_Duration', 'GoogleAnalytics_BounceRate', 'GoogleAnalytics_PageValue', 'Product_Pages', 'November', 'Other_Month', 'Phone', 'Other_Country', 'Portugal', 'Spain', 'Returner', 'Browser_2', 'Traffic_2'

Annex 9 – AdaBoost Feature Importance

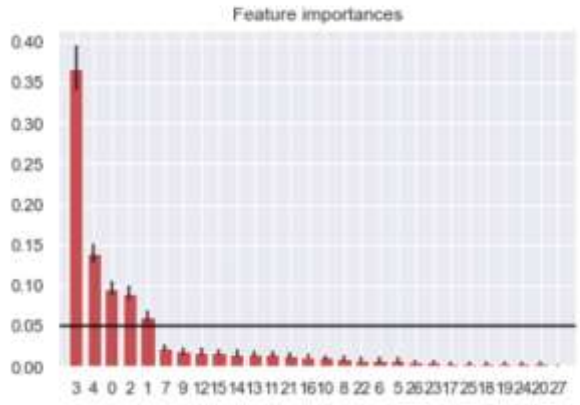


Table 6 – Cross Validation Scores

		LogisticRegression	DecisionTreeClassifier	Neural Networks	Gaussian NB	K Nearest Neighbors
Random Forest Features	**Training**	0.8090	**0.9901**	**0.8498**	0.7874	0.8494
	Validation	0.6471	**0.5495**	**0.6572**	0.5582	0.6512
Combination Features	**Training**	0.8079	**0.9141**	**0.8623**	0.7261	0.8479
	Validation	0.5979	**0.5530**	**0.6234**	0.5630	0.6646
AdaBoost Features	**Training**	0.8116	**0.9921**	**0.8476**	0.7128	0.8327
	Validation	0.6462	**0.5515**	**0.6571**	0.5900	0.6162
Lasso Regression Features	**Training**	0.8168	**0.9975**	**0.8765**	0.7160	0.8410
	Validation	0.6260	**0.5592**	**0.6331**	0.5897	0.6058
RFE Features	**Training**	0.8153	**0.9955**	**0.8634**	0.7111	0.8416
	Validation	0.6230	**0.5718**	**0.6407**	0.5884	0.6090

Table 7 – Cross Validation Scores (after hyper-parameter tunning)

		DecisionTreeClassifier	Neural Networks
Random Forest Features	**Training**	0.8573	0.8456
	Validation	0.6561	0.6564
Combination Features	**Training**	0.8549	0.8527
	Validation	0.6530	0.6525
AdaBoost Features	**Training**	0.8631	0.8744
	Validation	0.6618	0.6313
Lasso Regression Features	**Training**	0.8699	0.8344
	Validation	0.6727	0.6138
RFE Features	**Training**	0.8672	0.8833
	Validation	0.6753	0.6486

Table 8 – Cross Validation Scores for Ensemble Classifiers

		Voting	Stacking	AdaBoost	Bagging	Gradient Boosting Classifier	Random Forest Classifier	Histogram-based Gradient Boosting
Random Forest Features	**Training**	0.8492	0.8599	0.8911	0.8458	**0.8839**	**0.9901**	**0.8792**
	Validation	0.6229	0.6580	0.6551	0.6562	**0.6506**	**0.6239**	**0.6727**
Combination Features	**Training**	0.8520	0.8554	0.8662	0.8533	**0.8660**	**0.9143**	**0.8424**
	Validation	0.6669	0.6572	0.6410	0.6540	**0.6538**	**0.5485**	**0.6588**
AdaBoost Features	**Training**	0.8661	0.8728	0.9024	0.8117	**0.8968**	**0.9921**	**0.8966**
	Validation	0.6651	0.6557	0.6599	0.6387	**0.6678**	**0.6423**	**0.6941**
Lasso Regression Features	**Training**	0.8904	0.9334	0.9295	0.9389	**0.9201**	**0.9975**	**0.9181**
	Validation	0.6592	0.6363	0.6375	0.6290	**0.6857**	**0.6426**	**0.6993**
RFE Features	**Training**	0.8781	0.8799	0.9249	0.8822	**0.9126**	**0.9955**	**0.9399**
	Validation	0.6606	0.6701	0.6411	0.6551	**0.6851**	**0.6473**	**0.7823**

Table 9 – Ensemble Cross Validation Scores (after hyper-parameter tunning)

		Gradient Boosting Classifier	Histogram-Based Gradient Boosting	Random Forest
Random Forest Features	Training	0.7227	0.8467	0.8469
	Validation	0.6861	0.6661	0.6715
Combination Features	Training	0.7505	0.8503	0.7506
	Validation	0.6642	0.6605	0.6642
AdaBoost Features	Training	0.7536	0.8565	0.7536
	Validation	0.7738	0.6661	0.7739
Lasso Regression Features	Training	0.7611	0.8680	0.8626
	Validation	0.6982	0.6707	0.6588
RFE Features	Training	0.7612	0.8692	0.8553
	Validation	0.7112	0.6650	0.6663

Table 10 – Final Scores for Best Model in each set of features

Best Classifer for Each Set of Features			Score	Recall	Threshold *
Random Forest Features	GradientBoostingClassifier (1)	Training	0.7130	0.7193	0.347
		Validation	0.6627		
Combination Features	GradientBoostingClassifier (2)	Training	0.6895	0.6903	0.325
		Validation	0.6475		
AdaBoost Features	Histogram-based Gradient Boosting (3)	Training	0.6662	0.7870	0.5
		Validation	0.6429		
Lasso Regression Features	Histogram-based Gradient Boosting (4)	Training	0.6872	0.7548	0.557
		Validation	0.6610		
RFE Features	Histogram-based Gradient Boosting (5)	Training	0.6865	0.6903	0.591
		Validation	0.6645		

*Defined threshold for cutoff value of predicting class label ‘1’

(1) GradientBoostingClassifier(criterion='mse', learning_rate=0.05, loss='exponential', max_depth=4, max_leaf_nodes=12, random_state=1, warm_start=True)

(2) GradientBoostingClassifier(criterion='mse', learning_rate=0.05, loss='exponential', max_depth=5, max_features='log2', max_leaf_nodes=8, random_state=1, warm_start=True)

(3) HistGradientBoostingClassifier(learning_rate=0.03, max_depth=2, max_leaf_nodes=12, random_state=1, scoring='f1')

(4) HistGradientBoostingClassifier(learning_rate=0.03, max_depth=3, max_leaf_nodes=12, random_state=1, scoring='f1')

(5) HistGradientBoostingClassifier(learning_rate=0.03, max_depth=4, max_leaf_nodes=8, random_state=1, scoring='f1')

7. Webgraphy

https://scikit-learn.org/stable/auto_examples/model_selection/plot_successive_halving_heatmap.html

SMOTE for Imbalanced Classification with Python (machinelearningmastery.com)

machine learning - How to perform SMOTE with cross validation in sklearn in python - Stack Overflow

Imbalanced Classification in Python: SMOTE-Tomek Links Method | by Raden Aurelius Andhika Viadinugroho | Towards Data Science

Histogram-Based Gradient Boosting Ensembles in Python (machinelearningmastery.com)