NOVA IMS Information Management School

Master in Data Science & Advanced Analytics

Machine Learning

Techscape Project Report

Alice Vale R20181074
Eva Ferrer R20181110
Rafael Sequeira R20181128
Raquel Sousa R20181102
Rogério Paulo M20210597

Group 20

December 2021

Contents

I. Data Exploration

The project began by trying to better understand our target's behavior. By plotting the count of *Buy*, it was latent an asymmetry in the dataset, with a considerably higher number of *0's* than *1's*. On top of that, we performed descriptive statistics on the original variables by the *Buy* label, we graphically inspected how the target was distributed in the training dataset, with the help of bar plots. For each numeric variable, we analyzed their histogram and boxplot to detect possible patterns and hidden categorical variables, such as *Type of Traffic* and *Browser*. Moreover, we performed a pairwise relation hue and plotted the violin and density plots for each variable. Finally, we performed a heatmap to assess the correlations among themselves and the target. For the categorical variables, we plotted their absolute frequencies and evaluated how *Buy's* mean for each category.

II. Coherence Check

To guarantee that our data had no incongruences that could affect and bias the classification performance of our models, we performed coherence check. Firstly, we considered to be illogical that a user has visited a certain page type (Product, Account or FAQ) but there is no time spent in that type of page, that is, the duration of the session for that page is zero. This indicates that there might be visitors who visited a page type and the time spent there was not accounted for, or they did not visit the page and the count of visits is wrong. There were 877 observations that met this condition therefore, instead of removing all the observations from the dataset, which would lead to major information loss, we had two options: we could either accept the existing data and assume that the session durations were so low that the values were rounded to zero or we could alter the data, so the relations between variables made as much sense as possible. We chose to apply the second option and alter the value of the number of visits to zero, if the duration in that page type was also zero. We are aware that this imputation could alter the importance and effect of the variables in the classification process, however, we believe that it is important to have coherent and sensible relationships between the variables.

Secondly, we assumed that if the visitor made a purchase (*Buy = 1*) then they had to mandatorily visit a *Product Page,* as there cannot be purchases made without entering the catalogue page and selecting the wanted product or service. A total of 10 observations were removed from the dataset, so to guarantee the integrity and truthfulness of our data. Because this value is very small, we continued the data treatment process in no doubt that we were respecting the 3% rule of observation removal. Finally, it is important to mention that the observations were eliminated using a function that contained the removal clauses created by us.

III. Outliers & Missing Values

An important step in data preparation is to verify the existence of possible outliers, which can affect the classification performance of the algorithms. The variables *AccountMng_Duration, FAQ_Duration, Product_Pages, Product_Duration* and *GoogleAnalytics_PageValue* showed evident signs of having outliers for their high *standard deviation* and large difference between the *quantiles,* during the outlier's inspection stage. The histogram and boxplot showed that most variables presented an accentuated *right-skewed* distribution of their values, which indicates that the majority of the observations are centered around lower values and there might be outliers biasing the appearance of the plot. In fact, the boxplots validated the presence of multiple outliers and those values were later used to create the removal clauses for the biasing observations. A function was created to facilitate the outlier removal, which combined an instance of a previously created function, that applies *IsolationForest* to the dataset in order to identify outliers, and the manual condition clauses for outlier removal, created by us.

In the end, 92 observations are removed, which corresponds to less than 1% of the observations, thus respecting the 3% rule for observation removal. It is also important to mention that during the initial exploration phase we assessed that none of the variables in the dataset contained missing values, so no further treatment was required regarding them

IV. Feature Engineering

In total, over 50 new variables were created to maximize the performance of the prediction of our target, *Buy*. A detailed explanation of what variables were created, their meaning and how they were created can be consulted in the Annexes. Most variables created resulted from visually analyzing each original variable individually and trying to understand if it made sense to group values together or even perform joint statistics on multiple variables due to common behaviors.

In addition, there were created six Principal Components originated from the original metric features and its standardized data. (See more in the Annexes) To evaluate their explained variance regarding the initial variables, PC's loadings were accessed, and it was considered as a positive correlation, loadings above the threshold 0.45; and as a negative correlation, loadings below -0.45. That was very helpful to understand which original variables were better explained by which PC.

V. Feature Selection

For the selection of features to be used on the training of our predictive models, we decided to merge the results of 5 different algorithms. All of these had the objective of outputting sets of 10 variables, to then be compared. All features used were numerical and this study was carried out with the standardized original dataset, prior to the removal of outliers but without incoherent data.

The first algorithm taken into consideration was *Mutual Information*, proper when working with a discrete target, which measures the mutual dependency between variables.

In this case, the dependency was calculated between the independent variables and the target, therefore, a higher dependency value was desirable. *Recursive Feature Elimination*, a commonly used wrapper method, was then used, with a *Random Forest* as estimator. Here, features are removed at each iteration, while checking for the overall performance of the model. The lowest the position in the ranking, the higher the variables importance in explaining the problem in hands.

Thirdly, and since this was a classification problem with only numerical features, the usage of *ANOVA f-test* was permitted. Here, similar to what happens in mutual information, the features that show little dependency with the target are eliminated. An embedded method, *Lasso* regression, was also used. This is known for its high capability for the selection of features. If the coefficient assigned to variables is equal to zero, they are not important to explain the target, and therefore should be discarded. The last algorithm considered was a combination of *decision trees*, each one using a different criterion of quality, *gini* or *entropy*. The lower these values, the better the feature, since they both are measures of impurity or misclassification.

These methods provided different sets of result variables, so we assigned a binary flag to each feature, where 1 means chosen by the algorithm and 0 the inverse, and summed their scores. Combining this result with an analysis of the correlation matrix, we ended up constructing 5 different sets of features to be used later. The *Principal Components* were also gathered in a subset so to take advantage of their organic orthogonality. After this, a preliminary study was performed, with the use of the standardized data without outliers, where we concluded that the subset 4 had in general the best *f1_score* results for most classifiers, while subsets 3 and 5 were generally the less promising ones.

VI. Scaling & Data Partitioning

As it was our objective to try out some predictive models which required data to be scaled, we opted for the creation of four datasets, since different models could show distinct behaviors with each scaling procedure. We applied *Standard Scaler*, *MinMaxScaler* with a [-1,1] range and *RobustScaler*, each identified by a suffix in the end of the name of the dataset, *stand*, *norm* and *robust* respectively.

These changes, combined with the treatment or not of outliers in the dataset resulted in eight different versions of the initial dataset to be tested along the study. When performing predictions for the test set, this will also go under the scaling process, depending on how the model itself was trained.

The last procedure before moving forward to the predictive analysis is to split the dataset into a training section, which will be used to teach the behaviors of our data to the models; and a validation dataset, which will serve as a mean of evaluating the performance of the prediction on unseen data as well as to check for the existence of overfitting. We decided to go for an 80/20 split, since the number of observations was not extensive, and it was imperative to ensure enough data was available for training.

We also guaranteed stratification on the target variable, so to keep the same proportion of *0's* and *1's* in the datasets created through the split. This is extremely relevant since this dataset revealed itself to be deeply imbalanced, with approximately 85% of non-buying customers.

Having achieved the predictive modelling phase, we settled for the use of 13 different algorithms namely: *Gaussian Naïve Bayes, Logistic Regression, K-Nearest Neighbors, Decision Trees, Neural Networks, Passive-Aggressive Classifier, Quadratic Discriminant Analysis, Support Vector Machines, Random Forest, Bagging, Voting, Stacking, AdaBoost & Gradient Boost Classifier*.

VII. Predictive Modelling Stage

As the target variable is categorical, and more specifically binary, the models assessed were Classifiers. For all the models tested, the parameters were tuned with the help of *GridSearch()* or with an adaptation of a search function for the *f1* score. When beneficial for testing purposes, graphics were computed to compare different possible parameters' values, accounting either for the highest *f1* score or for the lowest *oob* error. Two functions were created to have a deeper understanding of the resulted scores: Model Test and *Stratified K Fold*. The first calculates the predictions and returns the metrics; the latter, applies the *K fold* ten times, and returns the average of those metrics. Both functions were repeatedly called throughout the *Predictive Modelling* stage. When possible, a *random state* parameter was defined to be able to replicate the results. From now on, the models will be discussed accordingly to their complexity, ranging from the simplest to the most challenging one.

Logistic Regression

Despite accessing its results for different values of the parameter *solver,* the model shown consistently high overfitting patterns and very low scores. For that reason, the group decision was not to hyperfocus on this model and move on with potentially better ones. Anyhow, the best model tested was *LogisticRegression(random_state=10,solver='lbfgs')* for the robust, with outliers dataset, accounting for the variables in the third subset.

Naïve Bayes

The data being scaled or not is irrelevant, since Naïve bayes is based on probability rather than distance, the only different behavior here is between the datasets with or without outliers. There were no parameters to tune, except for *var_smoothing*, which did not affect the final *f1 score* of the models considerably, only their mean accuracies. Using the data in the original scale with and without outliers and comparing all subsets created: It showed always some underfitting, and the best combination were using the subset 4 with original data.

K-Nearest Neighbors

The starting point consisted of testing for all the subsets and scaling options with the *KNN* default parameters. After finding the optimal number of k neighbors for each combination of subset and dataset, the models with the better *f1* scores were further fine-tuned for other parameters.

The parameters that affected the results the most were the *metric*, *weights* and *algorithm*. Nonetheless, *metric* was the only one that positively impacted the score. The best result returned was: KNeighborsClassifier(n_neighbors=19, metric = 'manhattan') for the original dataset with standardized data, accounting for the variables in subset 4. To improve the score, a threshold was adjusted to 0.3157, as the precision recall curve suggested.

Decision Trees

The parameters in Decision Trees do not improve the scores, only attenuate the overfitting. The following parameters were tested: *criterion, splitter, max_depth, min_samples_split, min_samples_leaf, min_weight_faction_leaf, max_features, min_impurity_decrease, class_weight* and *ccp_alpha*. Keeping in mind, this algorithm is overall robust to outliers, it would not deeply affect the end results testing for with or without outliers. Two different approaches were taken into account: one for the *ccp_alpha* parameter, and the other one for the remaining parameters. The first, evaluating the relationship between the total impurity and the effective alpha, then the *f1* scores were accessed comparing with the alphas. Resulting in the best model being *DecisionTreeClassifier(ccp_alpha~0.001142)* for the standardized data, with the variables of subset 1; and the optimal threshold 0.286 provided by the precision recall curve. The second, showed the best scores with *DecisionTreeClassifier(criterion='entropy', splitter='best', min_samples_split=60, min_weight_fraction_leaf=0.15)* for standardized data, accounting for the third subset variables. The optimal threshold in this case was 0.5566.

Neural Networks

Testing for the Classifier Multi-Layer Perceptron was the most challenging and time-consuming task. Our method was beginning with only default parameters and testing for all the scales and subsets available, and accounting for our dataset with and without outliers. Several parameters were later successively tuned and incorporated in the model, and the *f1* scores for those intermediate models were computed. The elements tuned were: *hidden_layer_sizes* (Single and Multiple combinations of layers)*, activation, solver, alpha, batch_size, learning_rate, learning_rate_init, max_iter,random_state, tol, early_stopping, beta_1, beta_2,epsilon, n_iter_no_change*. In addition, to improve the score, the optimal threshold was evaluated with the precision recall curve. It is important to highlight that some parameters combinations proved to be more successful than other, namely: *tanh* as the activation and *adam* as the solver; *logistic* as the activation and *sgd* as the solver. To conclude, the best model achieved had the following specification: *MLPClassifier(activation = 'tanh', solver = 'adam', hidden_layer_sizes=(8), learning_rate_init = 0.008, batch_size = 150)* fitted to the robust, without outliers datasets, for the variables of subset 4; the optimal threshold found was 0.52.

Passive-Aggressive Classifier

The robust data is the only scaling method that provided good results from the get-go. Firstly, a default model was tested and returned impressive results, in average, the *f1* score rounded 0.67. The parameters were then adjusted, namely: *C,fit_intercept, max_iter, tol, early_stopping, n_iter_no_change, loss* and *class_weight*.

The final best solution was *PassiveAggressiveClassifier(random_state=10, loss='squared_hinge', C=0.5, n_iter_no_change=7),* and resulted in a slightly improvement from the default model. This holds true for the datasets without outliers, having in consideration the variables from the first subset.

Quadratic Discriminant Analysis

The only parameter of this model, *reg_param*, was fine-tuned especially for the subsets four and five, since along the development of various models, these were the subsets that overall returned the best results, therefore, the objective was to, later on, perform Ensemble models with them. The best results were provided by *QuadraticDiscriminantAnalysis(reg_param=0.99)*, considering the robust data without outliers and having in mind the variables from subset 4. The optimal threshold for this model was 0.0117.

Support-Vector Machine

This model requires scaling in order to correctly access their features' importance. The parameters tested were *C, kernel, degree, gamma* and *probability*. The best solution found was *SVC(C=0.4, probability=True)* with a threshold of 0.0744; using the standardized data from the dataset with outliers. Once again, the most tested subsets were the forth and fifth, and the best results originated from the subset number 4.

Random Forest Classifier

This model has the same parameters as a Decision Tree, however they were once again tuned for this specific ensemble model, the main parameters tuned were *max_depth, max_features, min_samples_split, min_samples_leaf, n_estimatorsa* and *oob_score.* The best result was provided by *RandomForestClassifier(max_features=6, min_samples_split=100, n_estimators=10,oob_score=True, random_state=0*), for standardized data without outliers, for the variables in subset four. The optimal threshold in this case was 0.4539.

Bagging Classifier

This model can either perform for *base_estimator* knn or decision trees, after a graphical inspection of the variance of *f1* score, the decision trees showed consistently better performances. Several other parameters were subject to either an individual or a collective analysis, namely *bootstrap, max_samples, max_features and n_estimators*. Overall, this algorithm showed latent overfitting issues, which denigrated its trustworthiness. To overcome this problem, the *Random Forest Classifier* was used as the base estimator, to find out whether it provided better scores. In fact, it did with the following model *BaggingClassifier(base_estimator=RandomForestClassifier(n_estimators=95,max_depth=4,max_features = 4), n_estimators=200,oob_score=0.1)* for a threshold of 0.4154. The subset used was the fourth one, for standardized data with the original dataset.

AdaBoost

The parameters tuned for were *base_estimator, max_depth, n_estimators, learning_rate and algorithm*.

The *base_estimators* evaluated were both the *Decision Tree* and the *Logistic Regression*, resulting in better performances for the DT. From then on, the other parameters were adjusted based on the DT. The best model was *AdaBoostClassifier(n_estimators=12, learning_rate=0.05)* for the robust scaled data of the dataset without outliers, considering the subset four variables. Finally, the threshold was defined as 0.3352.

Gradient Boost

The parameters tested were *loss, max_depth, n_estimators, learning_rate, max_features and criterion*. Overall, the loss defined as exponential and the *max_features* as sqrt returned the best results. After individually computing the scores for each parameter, to minimize the overfitting, the best model was *GradientBoostingClassifier(learning_rate= 1.25, loss='exponential', max_depth=2, max_features='sqrt', n_estimators=30)* for the standardized data of the original dataset, accounting for the variables of the first subset. The defined threshold was 0.3352.

Stacking Classifier

Since the fourth subset for the standardized dataset with outliers turned out as the best model for different algorithms, it was decided to test those same models with Stacking, resulting in an improvement of the individually assessed scores. The models stacked were the *SVM*, the *KNN* and the *NN*, and the parameters used can be consulted in the respective paragraph. Other common subsets' models were tested, but their score did not show any improvements with any combination of models.

Voting Classifier

Taking into account the Staking results, we decided to perform Voting on the same models and data as we did in the previous step, evaluating for the parameter *voting* as hard and soft as well as adjusting the *weights* given to each classifier. At the end, the Voting scores outperformed the Staking ones, culminating in the best model being an ensemble of the *Bagging*, the *SVM*, the *KNN* and the *NN*. The reasoning behind this may be the fact that for averagely good models, the Voting returns better performances, while the Staking works the best with already outstanding models.

Multi-Modal Classification

After performing all the above models, we decided to drastically change the approach, since the results so far were behind our initial expectations. Firstly, the dataset excluding outliers was split into two, depending on a natural division of our dataset, based on the values of the variable *Google Analytics' Bounce Rate*. We have also tried to perform this method with the variable *Google Analytics' Page Value*, nevertheless, it did not resulted in as good results. Consequently, now there was a dataset for bounce rates equal to zero, and one for bounce rates higher than zero. After that, the training and validation slip was performed for each of the two new datasets, and such data was standardized. From the beginning, our strategy included individually modelling each dataset with a *Random Forest Classifier*, with appropriate and differentiated parameters, considering what was the best for only that portion of data since, throughout our modelling stage, it showed particularly favorable results and it was a well-known model for us to tune-in any parameter, if needed.

On top of that, all the fitted models were subject to the precision recall curve, to define the optimal threshold to maximize the *f1 score*. Moreover, two new variables were created, *Total Value* and *Value per second*. These variables were only created and, consequently, used whilst this very specific modelling technique.

<u>Model with the 1st dataset (Bounce Rate=0)</u>: An *RFE* analysis was performed to select the potentially best features. Therefore, a new subset of seven variables was created for further use. To assess how to reach the most similar value for the original segment of 0's and 1's of the target for this dataset, 0.18 of 1's, we tried two different approaches: with and without using *SMOTE*. Since it is well-known that the proportion of zeros and ones in the target is unbalanced, the Borderline SMOTE function was applied to detect and generate new synthetic samples; later, the model was fitted to the Random Forrest considering this new re-sampling.

<u>Model with the 2nd dataset (Bounce Rate>0)</u>: The same steps were followed as for the first subset. Namely, evaluating the *RFE* results, creating a new subset of ten variables, modelling with and without previously applying *SMOTE*. The original balance of the target for this subset was 0.12, subsequently using *SMOTE* accomplished the best result, and moving on as the prediction for this sub-dataset.

The thresholds were adjusted to find the optimal value for keeping the same percentage of *1's* in the test dataset as in the training dataset. Finally, the full prediction consisted of the joining the two datasets together. This was, in fact, our best scoring model, achieving,*0.70769* on *Kaggle*. Nonetheless, since the *Kaggle* scores only account for about 30% of the tested data, we do not blindly rely on its results.

VIII. Conclusion – Further Discussing the Final Solution

Overall, some models showed poor results and tendencies to overfitting, namely the *KNN,* and, unlike what expected, the *Bagging Classifiers* with *decision tree* as base estimator*.* Despite *Multi-Modal Classification* being our best solution for dealing with this projects' main issue all along: the fact that the training data was unbalanced, which lead to biased estimations, since the *0's* were easier to predict than the *1's*; That was the only time we implemented a re-sampling technique, that might have been fruitful when applied to other modelling algorithms. However, this modelling technique have a huge downside: we cannot measure its overfitting.

Consequently, we decided to make the safest choice and accept as our final best model the Voting ensemble Classifier. Which combines the best out of the best models for *Bagging* (with *random forest classifier* as base estimator), *SVC*, *KNN* and *NN*. The importance of each of these models to the final prediction was tuned with weights, to maximize *NN's* relevance, since it overall had better performances. Our solution is a very consistent estimator, with almost zero overfitting.

IX. Annexes

Limitations

We split the original data into train and validation subsets after performing outlier detection, incoherence checking, features engineering and all the introductory data exploration analysis. That was a huge mistake, since it biases the analysis and consequent results of our modelling stages, compromising the credibility of the output scores and amplifying our chances of overfitting. We believe that our latent low scores and the overall predisposition of our models to overfitting can be, in part, explained by this poor choice.

Due to splitting the dataset after creating new variables, the features that rely on the computation of averages have bias results, because they include observations both from the train and validation sets. Therefore, it is likely that those features could perform better in predicting the target and, eventually, rank higher in the feature selection stage.

Another preoccupying limitation was our lack of early decision regarding both the best subset of features to use and scaling technique, despite the preliminary studies performed, which lead us to a path of too many (15) combinations to test in the modeling stage, total of 5 variable subsets and 3 scaling methods. That affected our view of the bigger picture and straightforwardness while testing each algorithm. Inversely, throughout the feature selection segment, we only used one version of our data, scaled with the Standard Scaler. Not having compared results in such an early stage may have drawn us back on evaluating the importance of some features.

Provided we had more time to perform more complex analysis with the *Multi-Modal*, we would have extended our range of classifiers tested with this approach, especially other ensemble models such as Voting or Stacking Classifiers. Furthermore, we would brainstorm possible ways of quantifying their validation score, since, at this point, it was only possible for us to assess each portion of the dataset's overfitting, and not a final, common one, resulting from joining both datasets together. We acknowledge that a lot of time was spent performing models with very little to nonpotential, instead of building more robust analysis aiming for quality over quantity of models tested.

Feature Engineering – New Features

Classification of Variables	Name	Description
Date related	Days since visit	Number of days between last visit and today
	Month	Month of the last visit
	Before Break	Dummy: "0" if the last visit occurred before April,1st 2020; "1", otherwise
Dummies related with Type of Traffic	Traffic 2Returner	A returner customer with type 2 of traffic
	Traffic 3Returner	A returner with types 3,14,12 and 13 of traffic
	Traffic New_Access	A new access with types 4,5,6, and 8 of traffic
	Traffic_2	Any visitor with type 2 of traffic
	Traffic_1&3	Visitors with types of traffic 1 and 3
	Traffic Cluster1	Visitors with types of traffic 5,7,8,10,11 and 15
Dummies related with Type of Visitor	Valuable Returners	Returners with a Google Analytics Page Value superior to 0
	Valuable New Access	New access visitors with a Google Analytics Page Value superior to 0
	New Access	Dummy variable that is 1 if the type of visitor is a new access, and 0 if is either Other or Returner
Dummies related with Google Analytics' Page Value	Value noAccPage	The Page Value is positive, and the user visited zero pages on account management
	Value noFAQPage	The Page Value is positive, and the user visited zero pages on FAQ
	High Value	The Page Value is higher than the mean of all Page Values.
Origins in Group Byes	Value Rate	Coefficient between Google Analytics' Page Value and the mean value of Google Analytics' Page Value per types of visitor & traffic
	Diff Bounce	Difference between the mean value of Google Analytics' Bounce Rate per types of visitor & traffic, and Google Analytics' Bounce Rate
Average Duration per Page	AccountMng Duration Mins; FAQ Duration Mins; Product Duration Mins	Total amount of time, in Minutes, spent by the user on: Account management pages, FAQ pages and Product pages, respectively.
	Avg AccountMng Duration perPages; Avg FAQ Duration perPages; Avg Product Duration perPages	The average duration, in Minutes, spent on each page visited related to account management, FAQ and Product, respectively.
Page-visit Information	Total Pages	The sum of the number of all pages visited, regardless of its category.
	Total Duration	The sum of the seconds spent on any page, regardless of its category.
	Product Page Rate	For product pages with visits that last more than 0 minutes, the coefficient of the number of product pages visited and the duration of the visit.
Value-visit Information	Value per Page	Coefficient between the Google Analytics' Page Value and the number of product pages visited
	Avg value per Product duration	Coefficient between the Google Analytics' Page Value and the duration in minutes of visits in product pages
	Value avg Product duration rate	Coefficient between the Google Analytics' Page Value and the average of duration in minutes of visits in product pages
Exit & Bounce Rate	Diff Exit Bounce	Difference between Google Analytics' Exit Rate and Bounce Rate
	Bounce by Exit Rate	For Google Analytics' Bounce Rate and Exit Rate values higher than zero, the coefficient of the values for bounce rate and exit rate.
One Hot Encoder	Android; Apple;Linux;Windows	Transforming nonmetric features into dummies, related with the original variable OS
	Brazil; France; Germany; Italy; Portugal; Other;…	Transforming nonmetric features into dummies, related with the original variable Country
	October; November; December; February¸ September;…	Transforming nonmetric features into dummies, related with the created feature Month
Variables created specifically for Multi-Modal Classification	Total Value	The product of Page Value and Product Pages.
	Value per Second	The Total Value divided by the Product Duration, in seconds.

Table 1 - New Features

Subsets of Variables

Subset1	Subset2	Subset3	Subset4	Subset5
AccountMng_Pages avg_value_PerProdDur Valuable_new_access Value_noFAQPage Value_noAccPage x2_11 Valuable_Returners Bounce_by_Exit_rate GoogleAnalytics_PageValue New_Access Traffic_1&3	AccountMng_Pages avg_value_PerProdDur Valuable_new_access Value_noFAQPage Value_noAccPage x2_11 Valuable_Returners Bounce_by_Exit_rate Value_avgProdDuration_rate New_Access Traffic_1&3	AccountMng_Pages avg_value_PerProdDur Valuable_new_access Value_noFAQPage Value_noAccPage x2_11 Valuable_Returners Bounce_by_Exit_rate GoogleAnalytics_PageValue Traffic_1&3	AccountMng_Pages Value_perPage Valuable_new_access Value_noFAQPage Value_noAccPage x2_11 Valuable_Returners diff_Exit_Bounce GoogleAnalytics_PageValue Traffic_1&3	AccountMng_Pages AccountMng_Duration_Mins FAQ_Duration_Mins Product_Duration_Mins x2_11 Valuable_Returners GoogleAnalytics_ExitRate GoogleAnalytics_PageValue FAQ_Pages New_Access

Table 2 - Original Subsets of Variables

New_Subset_1	New_Subset_2
AccountMng_Pages GoogleAnalytics_ExitRate Value_per_second Total_Value New_Access X2_11 Product_Duration_Mins	AccountMng_Pages AccountMng_Duration_Mins GoogleAnalytics_BounceRate Value_per_second Total_value Product_Pages Diff_Exit_Bounce X2_11 Value_noFAQPage New_Access

Table 3 - Subsets of Variables for Multi-Modal Predictions

Kaggle Scores of the best result for each Classifier

Model	Kaggle Score
Multi-Modal	0.70769
Voting	0.70050
Stacking	0.69696
MLP	0.69651
Random Forest	0.69072
Bagging	0.67980
Passive-Aggressive	0.67873
GradientBoost	0.67647

Model	Kaggle Score
SVM	0.67632
KNN	0.67605
QDA	0.66995
Decision Trees	0.66079
AdaBoost	0.65811
Naïve Bayes	0.64948
Logistic Regression	0.63157

Table 4 - Kaggle Scores

Final Solution: Parameters

VotingClassifier

I. Estimators:
   i. BaggingClassifier(base_estimator=RandomForestClassifier(n_estimators=95, max_depth = 4,max_features = 4), n_estimators = 200,oob_score = 0.1)
   ii. SVC(C=0.4, probability=True,random_state=10)
   iii. KNeighborsClassifier(n_neighbors = 19, metric = 'manhattan')
   iv. MLPClassifier(hidden_layer_sizes = (8),max_iter = 300, random_state = 10, learning_rate_init = 0.1, batch_size = 80, tol = 0.01)

II. n_jobs = -1
III. voting = 'soft'
IV. weights = [20,20,20,40]

- ✓ Fitted to training dataset with the presence of outliers, standardized data, considering the variables of the subset 4.
- ✓ A precision recall curve was evaluated, reaching an optimal threshold of 0.369622.
- ✓ The *f1-score* presented in the *jupyter notebook* was 0.711, leading to 0.7005 in *Kaggle*.

X. Self-Learning Algorithms

(In order of appearance in the Project)

1. Principal Components Analysis - PCA

PCA is a linear dimensionality reduction technique that consists of transforming the number of variables in the data into principal components (PC). In such a manner that the first PC accounts for the largest possible variance in the dataset, being a projection of the data points into a new axis that maximizes the variance - computed as the average of the squared distances from the projected points to the origin. The fact that PCA is an orthogonal linear transformation reflects into the fact that the second PC is uncorrelated with the first one, meaning that the new second axis created is perpendicular to the first; and it accounts for the next highest variance. That holds true for the total of *n* PC, being *n* the original number of variables, and *n* explaining a higher variance of the original dataset than *n-1*.

In addition, mathematically speaking, to understand how the initial variables are varying from the mean with respect to one another, and, therefore, to verify the relationships as correlations among the input variables, we compute the covariance matrix. With these matrix, one can define the eigenvalues and the eigenvectors. The directions of the new axes where there is the most variance are the eigenvectors; whereas the amount of variance carried in each PC are the eigenvalues, also known as the coefficients attached to each eigenvector.

To conclude, it is possible to reduce dimensionality without losing much diversity and information by discarding the components with low variance and considering the best components as the new variables. An important aspect to consider is the low interpretability of the PCs, since they do not have any real meaning by being a linear combination of the original variables.

<u>Practical Implementation:</u>

In our project, PCA was used to create a subset of PCs to later apply algorithms to. A function was defined by us in python that first standardized the data, and then created the components, suing a *sklearn* feature. Consequently, we assessed the loadings to better understand what initial variables were being mainly explained by each component.

From *sklearn.decomposition*, *PCA* was imported. It is worth mentioning the following attributes of such function, that were used to perform some posterior analysis on the output of PCA:

*components_*	The new axes in the feature space, projecting the directions of the maximum variance in the data.
*explained_variance_*	The eigenvalues of the components.
*explained_variance_ratio_*	Percentage of variance explained by each component
*singular_values_*	2-norms of the components variables in the lower-dimensional space.
*mean_*	The empirical mean per feature.
n_components	The estimated number of components.
n_features	The number of features in the training dataset.

2. Mutual Information Classification (MI)

Generally speaking, MI is calculated between two variables, and it measures the average reduction in uncertainty, calculated as the entropy, for one variable given a known value of the other variable. An important understanding regarding MI is that is always larger than or equal to zero. If zero, then the variables under study are independent, consequently, the larger the value of MI, the greater the relationship between these variables. Finally, MI is often used as a general form of correlation coefficient, measuring the dependence between random variables.

Practical Implementation:

From *sklearn.feature_selection.mutual_info_classif,* MI was imported. This function estimates the MI based on nonparametric methods, namely by accessing the entropy estimation from the k-nearest neighbors' distances. In our project, we used this method as a step in the feature selection stage, to perform a ranking on the metric variables based on their MI value with the target. MI was only applied, since our target variable is discrete. We developed our own function to manually define how many variables we wanted to be presented in the ranking; The ten highest scores were displayed. The following parameters were used to compute the MI scores:

X	The feature matrix
y	The target matrix
n_neighbors	Number of neighbors used for the estimation for continuous variables. As usual, higher number of neighbors reduce the variance of the estimation but can introduce a bias.

3. f_classif

The *f_classif* method *is* part of the feature selection techniques available in *scikit learn* and it uses *ANOVA* to calculate the F-value for the imputed data. This statistical test is commonly utilized to compare the variance of the mean or median of different groups. The *f_classif* takes as parameters, *X* and *y*, a set of regressors and the target, respectively, and returns the *F-statistic* and the *p-value* related to each feature. The higher the statistic value, the more relevant the feature is considered.

4. Passive Aggressive Classifier

This classifier belongs to the family of passive-aggressive algorithms which are mainly used for large-scaled learning, being the most common example the work with social media data. It is classified as an online-learning algorithm, since the input data enters in a continuous form, and the model is updated concurrently, contrary to what happens with batch-learning where the training dataset is injected all at once. However, the model does not have a clear picture of the entirety of data and is somewhat biased by the order of presentation of observations, and these aspects can come as a disadvantage and affect its performance.

These algorithms have similarities with the Perceptron model however they do not require a learning rate to be set but instead they include a penalization parameter *C*.

Since the practical problem in question on this report was one of classification, the algorithm in fact used was Passive-Aggressive Classifier. On an abstract level, the functioning of this classifier goes as the following: for correct predictions it acts passively, accepting the outcome since it was correct and keeping it unchanged; for misclassified observations it acts aggressively, changing the weights based on the penalization set by the user. The larger the value of *C*, the more aggressively misclassifications will be handled. Additionally, it is important to consider that this classifier is only able to predict the class itself, and not the class probabilities, so it is not a candidate for use with some ensemble classifiers.

This algorithm is available through the *sklearn* library for python usage and grants the analyst the ability of fine-tuning several parameters. The ones tested out in this context were:

- *C*: as stated before, it is the regularization parameter and defines the maximum penalty given to the misclassified examples. The default value is 1.
- *early_stopping*: if the algorithm should stop early when the validation score has stopped improving enough. This validation data will be a stratified section of the training data introduced, that will be kept aside for scoring. The stopping will be triggered if the change in score is not at least the value chosen for *tol* for a specific number of maximum iterations set by the user (*n_iter_no_change).* The default is False.
- *loss*: the loss function to be used for the measurement of the penalty applied to a misclassification example. The ones used were ‘hinge’ or ‘squared_hinge’.
- *fit_intercept*: this parameter defines if the estimation should include an intercept or not. The default value is True.
- *Max_iter*: number of maximum epochs for training. Default is 1000.

5. Quadratic Discriminant Analysis (QDA)

QDA is a generative model, with a quadratic decision surface, which assumes that the class-conditional densities are normally distributed, meaning that each class follows a Gaussian distribution. Therefore, the QDA classifier fits a Gaussian density to each class, with a quadratic decision boundary. That is generated using Bayes' rule by fitting class conditional densities to the input data. Moreover, the class-specific prior is the proportion of data points that belong to the class, while the class-specific mean vector is the average of the input variables that belong to such class; lastly, there are an estimate of variance and covariance for each class.

Practical Implementation:

From *sklearn.qda*, *QDA* was imported. This classifier only has two possible definable parameters: *prior* and *reg_param*. We decided to tune *reg_param* in order to find its optimal value, that could potentiate the f1 score, since this parameter regularizes the calculus of the covariance estimate.

6. Voting Classifier

The *voting classifier* is part of a broad collection of ensemble methods available in the *scikit learn* library. The process behind this technique is to merge multiple machine learning algorithms, namely in the *classifier* category, and either use a majority vote or a soft vote to predict the labels of our target. This method allows for the following parameters to be defined and controlled by the user:

1. **Estimators**

   Takes a list which contains a tuple for each model to be combined in the *voting classifier*. Each tuple holds the name or other identifier of the model and the respective specification of it.
2. **Voting -** If no method is chosen, the *classifier* uses the default value for voting: *Hard*.

   Allows to define the type of voting method to use, which can be:
   - **Hard**: The predicted class label is the one that corresponds to the majority of the labels predicted by each model, that is, the label that was predicted the most by the models. In case of a tie between the number of predictions for different class labels, the *voting classifier* will choose the label based on ascending sort order.
   - **Soft**: The predicted class label is based on the sum of predicted probabilities. The probability of each class being predicted by each model is calculated and then an average of those probabilities is calculated, for each class. Finally, the *classifier* chooses the class label with the highest average probability.
3. **Weights**

   Takes an array of weights that allow the user to assign different levels of influence to the models inside the *voting classifier*. When weights are imputed, the probabilities of each class are calculated, multiplied by the weight given to each model and averaged. The default value of this parameter is '*Uniform*', meaning that all models have the same power over the prediction of the label.

4. **N_jobs**

   Specifies the maximum number of jobs to run. Usually, this parameter is set as *None.*

5. **Flatten_transform**

   Allows to alter the shape of the output and is only used when *voting* = *'soft'.* If set to *True* it returns a matrix.

6. **Verbose**

   Allows to retrieve the elapsed time of fitting at the end of the process, if set to *True.*

7. Synthetic Minority Oversampling Technique – SMOTE

After identifying a point from the minority class, the nearest neighbors are identified and then, it is added *x* times more synthetic observations, by creating points along the line that joins the initial point and the selected neighbors. Withal, there is an issue with this technique: when there are observations in the minority class outer from one another, so far that they are mixed with observations from the majority class. That can possibly unify both classes, by creating a line bridge in-between.

The ***Borderline SMOTE*** is believed to overcome this constrain. This was the approached chosen by us to apply in our project. First of all, this algorithm classifies the minority class observations, if all their neighbors are the majority class, then such observation is labeled as noise, and hence is ignored while creating synthetic points. In addition, points that have both majority and minority class points as neighbors, are paying extra attention to, since they are considered as border points, and the algorithm completely re-samples from these points.

From *imblearn.over_sampling*, *BorderlineSMOTE* was imported. The parameters used in the practical implementation of this algorithm were the following.

sampling_strategy	It can assume different data types. 1. If float, it expresses the desired ratio of the number of samples, after resampling, in the minority class over the number of samples in the majority class. 2. If string,it expresses what is the targeted class by the resampling, meaning what is the class in which the number of samples will be equalized. 3. If dictionary, it expresses the keys as the targeted classes and the values as the desired number of samples for each targeted class.
k_neighbors	It corresponds to the number of nearest neighbors to be used to construct synthetic samples.
m_neighbors	It corresponds to the number of nearest neighbors to be used to determine whether a minority sample is in danger or not.
kind	The type of SMOTE algorithm to use, either "borderline-1" or "borderline-2".