Techscape | E-commerce

Machine Learning Project

Masters in Data Science and Advanced Analytics, major in Data Science

AUTHORS

Inês Ribeiro, m20210595

Matias Neves, m20211000

Afonso Gonçalves, m20210607

Vasco Pombo, m20211301

José Dias, m20211009

Table of Contents

Table of figures

Abstract

This project is about the development of a model that predicts if a customer is going to buy the products of the company or not. The company is called TechScape, and their core business revolves around selling products that helps customers do a digital detox.

Our goal is to answer the question: “Which customers are more likely to buy our products?”.

We started by doing data pre-processing, feature selection, creating dummy variables (one hot encoding), balancing the dataset, scaling, feature engineering and testing different models, with different datasets and hyperparameters.

After we found the best performing model, and the best hyperparameters, this corresponded to an f1 score of 0.73096 on Kaggle (0.68 in the validation dataset).

Introduction

Techscape is a company that sells products such as meditation kits, books, stress balls, "dumb" phones, retreats, among others, with the goal of helping a customer do a digital detox.

Techscape was founded in the beginning of 2020, by five portuguese entrepreneurs, but came out of business in April 2020.

The company suffered a lot with the global pandemic in 2020, like most companies, and restarted their business in May 2020.

Given that the company already faced difficulties in the past, they really want to target the “right customers”. Therefore, the company hired us as data scientists to predict which customers have a high probability of buying their products (based on their online behaviour).

The datasets that we have at our disposal are mainly related to the customer online actions.

Our goal is to obtain the best possible performance on a test dataset, by implementing binary classification on 2300 unseen records. To implement this, several algorithms need to be tested and their performance assessed, in order to understand which is the best one, given our data.

Data Exploration

First of all, it is important to understand what the variables mean, check how many features we have and also if we have missing/ duplicated values.

After that, we had to analyse in detail the features we have, to check if all of them make sense for the classification problem we have, and also to distinguish the numerical variables from the categorical variables (and check how many categories exist for every categorical variable).

Coherence Checks

Before dropping any features, or start applying models, we needed to make sure that our data is coherent.

We did the following coherence checks:

- Check if the date does not fall out of range (we should only have dates with year equal to 2020);
- Check if the countries make sense, ie, if all of them have internet access/ if they have any internet restrictions;
- Check the minimum values of the features, because there are some features (especially related with pages visited or duration) in which the minimum cannot fall below zero;

Outliers

The next step in the analysis was checking for any outliers to have the best possible representation of our data. Firstly, we plot the boxplots (figure 1) for all metric features to better visualize the distributions, and manually checked the number of outliers for some specific features with reasonable thresholds. Then we used two different methods of calculating outliers, we set a manual filter for every metric feature and calculated the upper and lower limit using the Interquartile Range method. Our final outlier detection would be the intersection of these two methods but since this resulted in just 23 observations out of 9985, we decided to keep all the records.

Feature Engineering

To try to have the best dataset possible, we decided to create new features that seemed relevant for this classification problem.

New Feature #1 : Recency

The first thing we noticed right away, is that the feature Date does not seem to give us much information in itself, therefore, we needed to change it. The first feature we created based on Date was Recency. The goal of this feature is to tell us if the customers bought more recently or if the purchase was not made recently. It gives us the number of days that passed from the purchase.

New Feature #2 : Month

The second variable created out of Date was Month. It simply gives us the month corresponding to the Date. The reason we believed this feature might be important is because we did a small analysis of the cvs file (train dataset) and noticed that there are some months where the proportion of 1´s is significantly bigger than in other months. If there are significant differences in proportions between the months, the variable seems relevant.

New Feature #3 : Weekday

The third variable created based on the variable Date was weekday. We decided to create this variable because it makes sense that there might exist some differences in buyers purchasing patterns depending on the day of the week.

New Feature #4 : AccountMng_Duration_per_page

The next feature we decided to create AccountMng_Duration_per_page. This is the division between AccountMng_Duration and AccountMng_Pages and it gives us the amount of time spent in each page.

New Feature #5 : FAQ_Duration_per_page

The last feature we decided to create FAQ_Duration_per_page. This is the division between FAQ_Duration and FAQ_Pages and it gives us the amount of time spent in each page

Scaling

Before we procced with the feature selection, it is important to scale our data, in order to put the features in the same range (and avoid giving more importance to some features, in detriment of others). Since we did not remove outliers, we decided to use the RobustScaler to scale our data (the major difference between the Robust and MinMax scaler is that the Robust uses IQR instead of Min-Max, therefore is not as sensible to the presence of outliers).

Feature Selection

Metric features

To do feature selection of the metric features, we use 5 different methods: Correlation Matrix (as seen in figure 2), RFE, Lasso (as seen in figure 3), Sequential Forward Selection (SFS), Sequential Backward Selection (SBS) - Sequential feature selection algorithms are a family of greedy search algorithms that are used to reduce the initial feature dimensional space to a smaller feature dimensional subspace. The motivation behind feature selection algorithms is to automatically select a subset of features that is most relevant to the problem. We concluded that we should keep 7 out of the 12 numerical variables that we had in the beginning.

The numerical features that we decided to keep are AccountMng_Duration, Product_Pages, GoogleAnalytics_BounceRate, GoogleAnalytics_ExitRate, GoogleAnalytics_PageValue and Recency (as seen in figure 4).

Categorical features

For feature selection of the categorical features, we just used one method: Chi-square. We kept 4 out of the 7 categorical features: OS, Type_of_Traffic, Type_of_Visitor and Month (as seen in figure 5).

One Hot Encoding

Since we couldn´t work with categorical features as they are, we had to create dummy variables using the One Hot Encoding method, ie, is one method to convert categorical data to prepare it for an algorithm and get a better prediction. With it, we convert each categorical value into a new variable and assign a binary value of 1 or 0 to those columns.

After using this method, we ended up with 30 features in total.

Balancing Dataset

At this point, we noticed that we didn´t have a balanced dataset (we had significantly less 1´s than 0´s in our target). Since we were trying to predict the 1´s, we decided to check if balancing the dataset could improve the quality of our predictions, so we tried undersampling, oversampling, a mix of both under and oversampling and with the unbalanced dataset.

We are aware "the random oversampling may increase the likelihood of occurring overfitting, since it makes exact copies of the minority class examples. In this way, a symbolic classifier, for instance, might construct rules that are apparently accurate, but actually cover one replicated example".

We are also aware that "in random under-sampling (potentially), vast quantities of data are discarded. […] This can be highly problematic, as the loss of such data can make the decision boundary between minority and majority instances harder to learn, resulting in a loss in classification performance".

Undersampling:
The first type of balancing we did was undersampling. This reduces the number of observations with target equal to 0. We tried the following methods of undersampling:

- NearMiss (version=1)/NearMiss (version=2): Near Miss Undersampling methods remove entries from the majority class based on their distance from entries in the minority class. Said distance is usually Euclidean. There are three versions of Near Miss, and two were used in this project: version 1 and version 2. Version 1 removes entries with the lowest average distance to the three entries in the minority class closest to the majority class. Version 2 has the same evaluation process, but uses the three entries in the minority class furthest from the majority class as a reference instead;
- OneSidedSelection(): is an undersampling technique that combines Tomek Links and the Condensed Nearest Neighbor (CNN) Rule;
- NeighbourhoodCleaningRule(): is an undersampling technique that combines both the k Nearest Neighbor (KNN) to remove redundant examples and the Edited Nearest Neighbors (ENN) Rule to remove noisy or ambiguous examples;

- TomekLinks(): this method can be used to find desired samples of data from the majority class that is having the lowest Euclidean distance with the minority class data (i.e. the data from the majority class that is closest with the minority class data, thus make it ambiguous to distinct), and then remove it;
- CondensedNearestNeighbour(): is an undersampling technique that seeks a subset of a collection of samples that results in no loss in model performance, referred to as a minimal consistent set.

The NearMiss undersampling method was by far the worst, therefore, we didn't try to optimize any model with it.

Oversampling:

- SMOTE: is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together.

Oversampling and undersampling:

- SMOTEENN: Combines over and undersampling using SMOTE, to generate samples for the minority class and Edited Nearest Neighbours to remove noisy or ambiguous examples

Modeling

We used several models to try to achieve the best predictions. Here is a summary of our analysis:

Best Model

The best score was achieved when data was balanced with the undersampling technique CondensedNearestNeighbour(n_neighbors=1, random_state=15) and using the model Voting classifier with: HistGradientBoostingClassifier, RandomForestClassifier and GradientBoostingClassifier.

We used these 3 classifiers because HistGradientBoostingClassifier and GradientBoostingClassifier were the best performing models individually, and then we tried to add an extra model. We tried LogisticRegression, KNeighboursClassifer and RandomForestClassifier and best model out of these 3 was the RandomForestClassifers (it gave us the best score in the validation dataset).

To find the best parameters, we did a gridsearch of the models individually, and then used the models together in the voting classifier.

The voting classifier uses voting = 'soft', because it gave us a better score than hard voting.

The parameters used in the best model were:

- HistGradientBoostingClassifier(learning_rate=0.05, max_bins=100, max_depth=3,max_iter=150, max_leaf_nodes=35,min_samples_leaf=15, random_state=15)
- RandomForestClassifier(random_state=15)

- GradientBoostingClassifier(random_state=15,criterion = 'mae', learning_rate=0.075, loss='deviance', max_depth=8,max_features='sqrt',min_samples_leaf=0.1,min_impurity_split=0.42727272727272736,n_estimators=10,subsample=0.8)
- VotingClassifier(estimators = estimator, voting ='soft')

The parameters used in the HistGradientBoostingClassifier gridsearch were loss, learning_rate, max_iter, max_leaf_nodes, max_depth, min_samples_leaf and max_bins.

The parameters used in the RandomForestClassifier gridsearch were n_estimators, max_features, max_depth and criterion.

The parameters used in the GradientBoostingClassifier gridsearch were loss, learning_rate, min_samples_split, min_samples_leaf, max_depth, max_features, criterion, subsample, n_estimators.

All of the values for each parameter of each gridsearch were changed and adjusted after the best one of the initial search. This means we did several gridsearches for each model to achieve the best one.

Our best model got a Kaggle score of 0.73096.

Other Models

Initially, we decided to use the following models (with default parameters), to see which ones performed better.

The models we used were: GradientBoostingClassifier, HistGradientBoostingClassifier, AdaBoostClassifier, SVC, MLPClassifier, LogisticRegression, GaussianNB, ExtraTreesClassifier, KNeighborsClassifier, VotingClassifier, StackingClassifier, DecisionTreeClassifier, RandomForestClassifier and SGDClassifier.

By far, our worst performing model was GaussianNB that scored between 0.29 and 0.40 in the validation dataset, therefore, we didn´t think it was worth it to optimize it or submit it on Kaggle.

The best performing models were: VotingClassifier, GradientBoostingClassifier, HistGradientBoostingClassifier and RandomForestClassifier. We optimize all of these with a gridsearch and improved the scores.

Other models that also performed well were: SVC, StackingClassifier, KNeighborsClassifier and AdaBoostClassifier.

Some scores that stood out from the rest were:

- HistGradientBoostingClassifier with CondensedNearestNeighbour(n_neighbors=1) undersampling, that achieved a score of 0.72081 on Kaggle

- GradientBoostingClassifier with OneSidedSelection(n_neighbors=1, n_seeds_S=200) undersampling, that achieved a score of 0.71957 on Kaggle
- RandomForestClassifier with no undersampling or oversampling, and that achieved a score of 0.70588 on Kaggle

We noticed that the models did not perform as well with the oversampling methods and also with no balancing. The best balancing approach was clearly undersampling, especially CondensedNearestNeighbour(n_neighbors=1, random_state=15).

Best Model Explanation

Voting Classifier is a model that uses an ensemble of numerous models and predicts an output based on their highest probability of chosen class as the output.

It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate models and finding the accuracy for each one of them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.

Voting Classifier supports two types of votings:

- Hard Voting: The predicted output class is a class with the highest majority of votes, i.e., the class which had the highest probability of being predicted by each of the classifiers;
- Soft Voting: The output class is the prediction based on the average of probability given to that class. For example, given some input to three models, the prediction probability for class A = (0.75, 0.56, 0.40) and B = (0.25, 0.44, 0.60). So, the average for class A is 0.57 and B is 0.43, in this case the winner is clearly class A because it had the highest probability averaged by each classifier.

We used three classifiers inside the Soft Voting:

The GradientBoostingClassifier relies on the assumption that the best possible next model, when combined with previous models, minimizes the overall prediction error. Gradient boosting involves three elements: a loss function to be optimized, a weak learner to make predictions and an additive model to add weak learners to minimize the loss function.

The loss function used depends on the type of problem being solved. For us, the best one out of the gridsearch was ‘deviance’. The weak learners used are Decision trees, and the gradient descent procedure is used to minimize the loss when adding trees, acting as the additive model.

The HistGradientBoostingClassifier is part of the same class of ensemble learning algorithms as the GradientBoostingClassifier since it works the same way (each tree model added to the ensemble attempts to correct the prediction errors made by the tree models already present in the ensemble), but is much faster because the ensemble can be dramatically accelerated by discretizing (binning) the continuous input variables to a few hundred unique values. Gradient boosting ensembles that implement this technique to transform input variables are referred to as histogram-based gradient boosting ensembles.

The RandomForestClassifier is a classification algorithm made up of multiple decision trees and uses both bagging and feature randomness to create an uncorrelated forest of decision trees. This feature randomness generates a random subset of features, that ensures a low correlation between decision trees. The difference between decision trees and random forests is that while decision trees consider all the possible feature splits, random forests only select a subset of those features.

Pain Points of our project

We are aware that our predictive model has some pain points, and therefore it can definitely be improved.

The first pain point that we want to mention are the outliers. We decided not to remove them, because there were only 23 observations considered outliers. However, removing those observations and using different scaling techniques could yield a higher score. Despite all that, we made sure that the scaling technique that we used makes sense from a theoretical perspective (we used RobustScaler because we did not remove outliers).

The second pain point we want to mention is related to two feature selection methods we have used. In the feature selection of metric features we used 5 different approaches. However, two out of those five were not used in an accurate manner. Those two are Sequential Forward Selection and Sequential Backward Selection. We used scoring as accuracy instead of using the f1 score (the score of our model). If we used f1 score in both methods, Sequential Forward Selection would not change the selected features, however in Sequential Backward Selection the selected features would change, and that could impact the final decision, if we keep a feature or if we drop it.

Those are the two points that we think can be improved in our project. Even though we think that it would be interesting to fix those two pain points, we also think the gains (if they exist) in the final score would only be marginal (given that our models are optimized with gridsearch, the outliers are only 23 out of 9999 observations and only one out of the five feature selection methods is not optimal).

Bibliography

(n.d.). Retrieved from https://scikit-learn.org/stable/supervised_learning.html

(n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html

(n.d.). Retrieved from https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/

(n.d.). Retrieved from https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

(n.d.). Retrieved from https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

(n.d.). Retrieved from https://www.ibm.com/cloud/learn/random-forest

(n.d.). Retrieved from https://machinelearningmastery.com/histogram-based-gradient-boosting-ensembles/

(n.d.). Retrieved from https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

(n.d.). Retrieved from https://www.geeksforgeeks.org/ml-voting-classifier-using-sklearn/

Fernández, A., García, S., Galar, M., Prati, R. C., & Krawczyk, B. (2018). *Learning from Imbalanced Data Sets.* Springer International Publishing.

Haibo He, Y. M. (2013). *Imbalanced Learning: Foundations, Algorithms, and Applications.* Wiley-IEEE Press.

Appendix

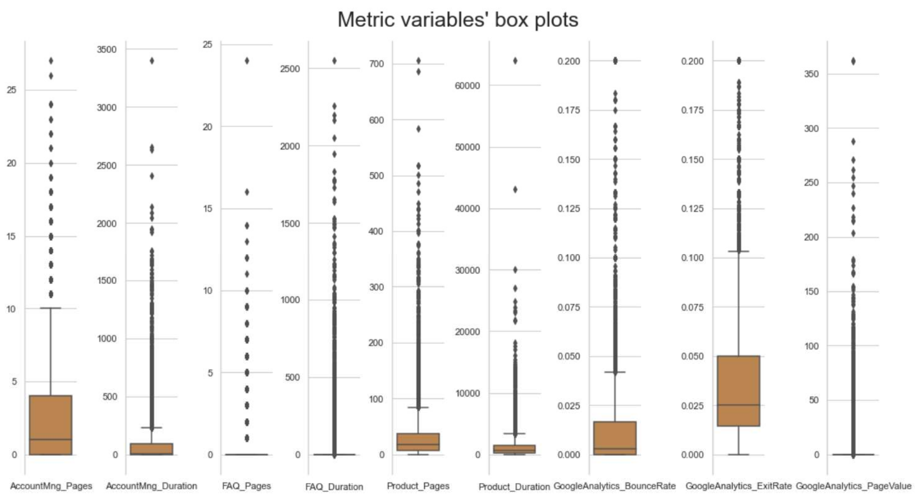
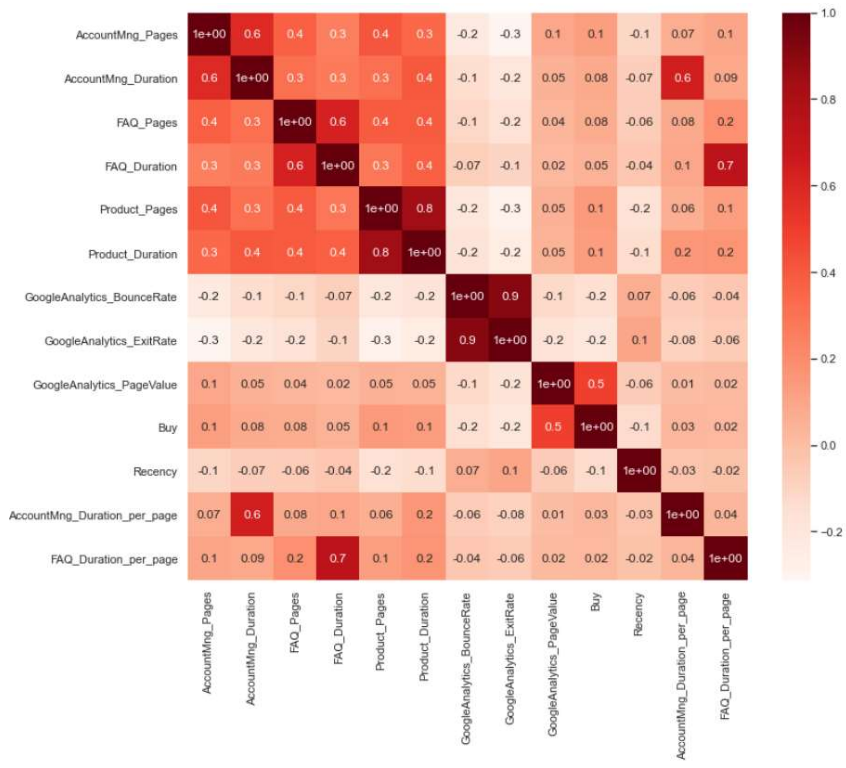


Figure 1: Metric variables' boxplots



	AccountMng_Pages	AccountMng_Duration	FAQ_Pages	FAQ_Duration	Product_Pages	Product_Duration	GoogleAnalytics_BounceRate	GoogleAnalytics_ExitRate	GoogleAnalytics_PageValue	Buy	Recency	AccountMng_Duration_per_page	FAQ_Duration_per_page
AccountMng_Pages	1e+00	0.6	0.4	0.3	0.4	0.3	-0.2	-0.3	0.1	0.1	-0.1	0.07	0.1
AccountMng_Duration	0.6	1e+00	0.3	0.3	0.3	0.4	-0.1	-0.2	0.05	0.08	-0.07	0.6	0.09
FAQ_Pages	0.4	0.3	1e+00	0.6	0.4	0.4	-0.1	-0.2	0.04	0.08	-0.06	0.08	0.2
FAQ_Duration	0.3	0.3	0.6	1e+00	0.3	0.4	-0.07	-0.1	0.02	0.05	-0.04	0.1	0.7
Product_Pages	0.4	0.3	0.4	0.3	1e+00	0.8	-0.2	-0.3	0.05	0.1	-0.2	0.06	0.1
Product_Duration	0.3	0.4	0.4	0.4	0.8	1e+00	-0.2	-0.2	0.05	0.1	-0.1	0.2	0.2
GoogleAnalytics_BounceRate	-0.2	-0.1	-0.1	-0.07	-0.2	-0.2	1e+00	0.9	-0.1	-0.2	0.07	-0.06	-0.04
GoogleAnalytics_ExitRate	-0.3	-0.2	-0.2	-0.1	-0.3	-0.2	0.9	1e+00	-0.2	-0.2	0.1	-0.08	-0.06
GoogleAnalytics_PageValue	0.1	0.05	0.04	0.02	0.05	0.05	-0.1	-0.2	1e+00	0.5	-0.06	0.01	0.02
Buy	0.1	0.08	0.08	0.05	0.1	0.1	-0.2	-0.2	0.5	1e+00	-0.1	0.03	0.02
Recency	-0.1	-0.07	-0.06	-0.04	-0.2	-0.1	0.07	0.1	-0.06	-0.1	1e+00	-0.03	-0.02
AccountMng_Duration_per_page	0.07	0.6	0.08	0.1	0.06	0.2	-0.06	-0.08	0.01	0.03	-0.03	1e+00	0.04
FAQ_Duration_per_page	0.1	0.09	0.2	0.7	0.1	0.2	-0.04	-0.06	0.02	0.02	-0.02	0.04	1e+00

Figure 2: Correlation matrix

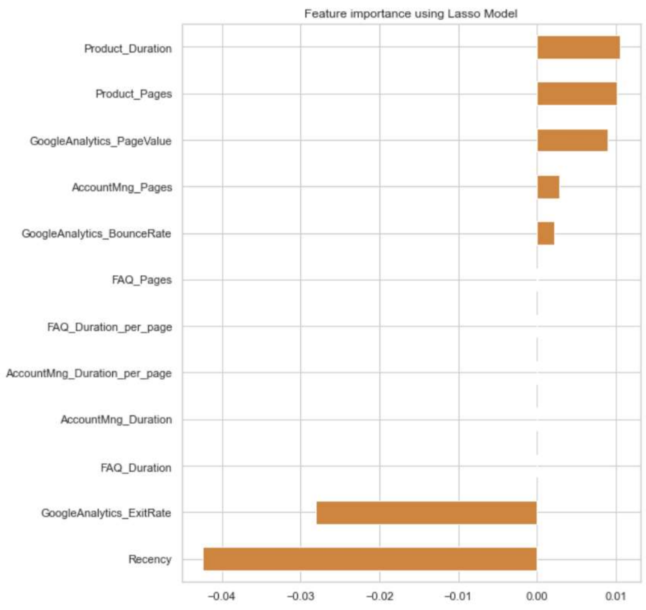
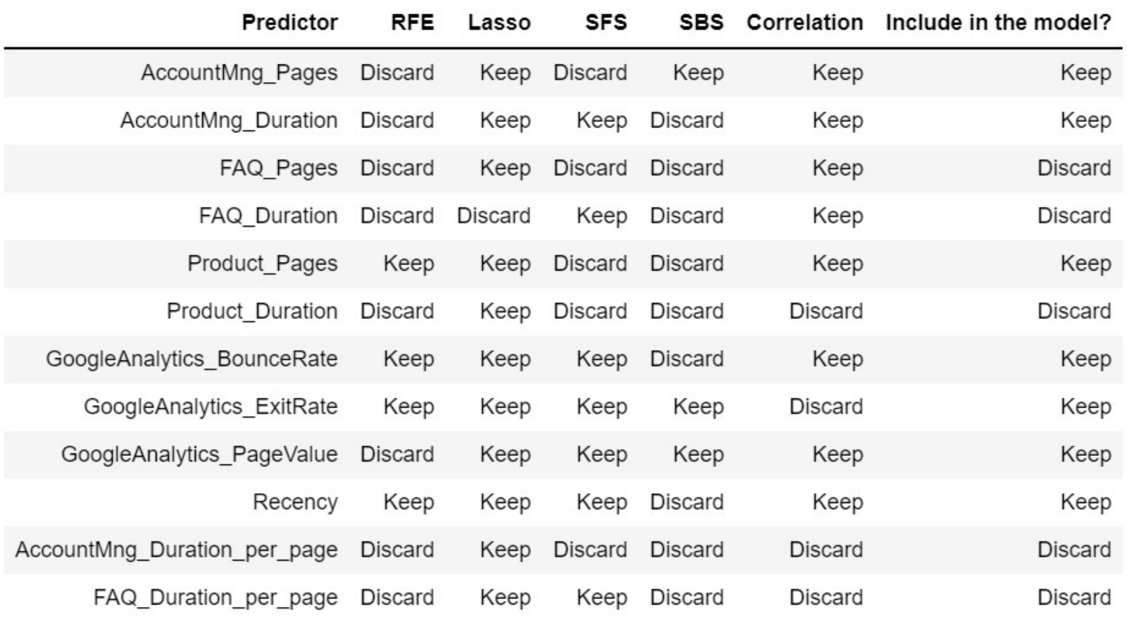
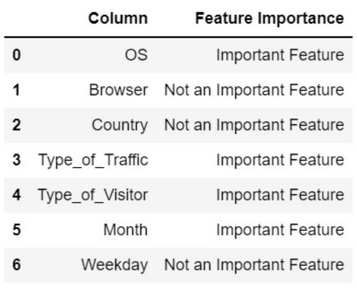


Figure 3: Feature importance using Lasso Model



Predictor	RFE	Lasso	SFS	SBS	Correlation	Include in the model?
AccountMng_Pages	Discard	Keep	Discard	Keep	Keep	Keep
AccountMng_Duration	Discard	Keep	Keep	Discard	Keep	Keep
FAQ_Pages	Discard	Keep	Discard	Discard	Keep	Discard
FAQ_Duration	Discard	Discard	Keep	Discard	Keep	Discard
Product_Pages	Keep	Keep	Discard	Discard	Keep	Keep
Product_Duration	Discard	Keep	Discard	Discard	Discard	Discard
GoogleAnalytics_BounceRate	Keep	Keep	Keep	Discard	Keep	Keep
GoogleAnalytics_ExitRate	Keep	Keep	Keep	Keep	Discard	Keep
GoogleAnalytics_PageValue	Discard	Keep	Keep	Keep	Keep	Keep
Recency	Keep	Keep	Keep	Discard	Keep	Keep
AccountMng_Duration_per_page	Discard	Keep	Discard	Discard	Discard	Discard
FAQ_Duration_per_page	Discard	Keep	Keep	Discard	Discard	Discard

Figure 4: Metric Feature Selection table



	Column	Feature Importance
0	OS	Important Feature
1	Browser	Not an Important Feature
2	Country	Not an Important Feature
3	Type_of_Traffic	Important Feature
4	Type_of_Visitor	Important Feature
5	Month	Important Feature
6	Weekday	Not an Important Feature

Figure 5: Categorical Feature Selection table