Universidade NOVA de Lisboa
NOVA Information Management School

Beatriz Vizoso
Ivan Jure Parać
Maria Almeida
Nuno Melo Bourbon
Stuart Gallina Ottersen

TECHSCAPE DATA EXPLORATION

Lisbon, 2021

Universidade NOVA de Lisboa

NOVA Information Management School

L I S B O N

Beatriz Vizoso
Ivan Jure Parać
Maria Almeida
Nuno Melo Bourbon
Stuart Gallina Ottersen

Programme: Data Science
Course: Machine Learning

TECHSCAPE DATA EXPLORATION

Mentors:

Roberto Henriques
Carina Albuquerque
Lara Oliveira

Lisbon, December 2021.

Beatriz Vizoso
Ivan Jure Parać
Maria Almeida
Nuno Melo Bourbon
Stuart Gallina Ottersen

Statement of Authenticity

Hereby We state that this document, Our project, is authentic, authored by Us, and that, for the purposes of writing it, We have not used any sources other than those allowed for this project. Ethically adequate and acceptable methods and techniques were used while preparing and writing this report.

Abstract

The present report has been prepared for TechScape in an effort to increase the company's sales by predicting the customers most likely to make a purchase based on their online behaviour. The original data, consisting of a train and a test datasets, was first explored before being preprocessed to be used for modelling. Multiple modelling algorithms were tested and evaluated mainly based on their F1 scores, to determine which would provide the best solution for the problem at hand. The final recommended approach is a hard Voting Classifier using, as estimators, a Random Forest Classifier, a HistGradient Boosting Classifier, and a Support Vector Classifier, based on the seven features selected during preprocessing.

Keywords: machine learning; python; big data; preprocessing; data analysis; classification; modeling; supervised learning

Table of Contents

1. Introduction . . . 1

2. Data Exploration and Understanding . . . 1

3. Data Preprocessing . . . 2

4. Modelling . . . 3

4.1. Linear Models . . . 4
4.2. Discriminant Analysis . . . 4
4.3. Support Vector Classification . . . 4
4.4. KNeighbors Classifier . . . 4
4.5. Naive Bayes . . . 5
4.6. Decision Tree classifiers . . . 5
4.7. Ensemble methods . . . 5
4.8. Semi-supervised learning . . . 5
4.9. Multi-layer Perceptron . . . 5
4.10. More ensembles . . . 5
4.10.1. Stacking . . . 5
4.10.2. Voting . . . 6
4.11. Conclusion . . . 6
4.12. Hyper-parameter optimisation . . . 6

5. Conclusion . . . 6

5.0.1. Improvements . . . 7
5.0.2. Deploying the model . . . 7

Bibliography . . . 9

List of Figures . . . 10

List of Listings . . . 11

1. Supplemental . . . 13

1.1. Hyper-parameter optimization . . . 13
1.2. Pipeline . . . 13
1.3. Scaling Methods . . . 13
1.4. Sampling Methods . . . 13
1.5. ANova . . . 14
1.6. Maximal Information Coefficient . . . 14
1.7. Models . . . 14

1.7.1. Discriminant analysis 14
1.7.2. BernoulliNB 14
1.7.3. Nearest centroids 14
1.7.4. Label Spreading and Label Propagation 14
1.8. Voting 15
1.8.1. Voting Model Performance 15

2. Figures 17

3. Listings 23
3.1. Listing 1. 23

1. Introduction

TechScape is a Portuguese digital detox company that sells its products, such as meditation kits, stress balls, and retreats, over an online store. Its main goal is to help its clients balance technology usage with other leisure activities. With the emergence of COVID-19, which prompted compulsory lock-downs and incentivized remote work, this task has never been more relevant.

The present report was prepared by a hired team of Data Scientists and analyses the online behavior of TechScape's customers in an attempt to predict potential buyers and ultimately boost the company's revenue. The predictions obtained are based on the information gathered on past buyers and non-buyers alike, such as page visits and Google Analytics metrics. Several processing and modeling methods were explored in an attempt to obtain the most robust predictions, as determined by the F1 score (a better metric than accuracy for an imbalanced dataset) upon their submission to Kaggle.

2. Data Exploration and Understanding

"*Data exploration is about efficiently extracting knowledge from data even if we do not know exactly what we are looking for.*" [1]

The train and test sets provided consisted of 9,999 and 2,300 registered accesses (i.e, rows) to TechScape's online store, respectively. The same 16 features were present in both datasets (including metric and categorical features), and are listed in ***Figure 1***. No typing mistakes or strange characters were found in object-type features (namely *OS*, *Country* and *Type_of_Visitor*) and none of the features had any missing values, but a quick inspection of summary statistics suggested the presence of outliers. Furthermore, most metric features had a skewed distribution with many zeros and a long right tail (***Figure 2***), which may be troublesome for a few modeling algorithms. This is a consequence of most accesses visiting few pages and spending a short amount of time in the online store. The training dataset also included a "Buy" feature, where the value 1 indicates a purchase. A purchase was made in fewer than 16% of the accesses, thus indicating an imbalanced dataset.

The main insights from preliminary data exploration are presented in ***Figure 3***. Most of the online store accesses originated from European countries, with Portugal and Spain representing approximately 58% of the total accesses in 2020. The biggest non-European market was Brazil (10%). Over 85% of all accesses were returning customers, suggesting extra effort should be made to divulge the website and acquire new clients. Most users accessed the website from their computers, as inferred from the number of accesses made from Windows and macOS devices (74%), which is a common preference for online purchases but may also indicate a poor user experience on the mobile version of the web page. Not much is known about *Browser* and *Type_of_Traffic* as they were both label-encoded from the get-go other than that there is a clearly dominant Browser and three dominant types of traffic. The distribution of buyers and non-buyers was similar across categories.

Interestingly, the number of accesses to the website was correlated with the severity of the pandemic situation in Portugal (***Figure 4***), spiking in March (compared to February) with the beginning of the lockdown, and dropping in the summer months with the lifting of the restrictions. The number of accesses then surged upwards in November and December, coinciding once again with stricter prevention measures.

3. Data Preprocessing

To prepare the data for modeling, we started by setting the *Access_ID* as an index and removing duplicate rows (14 in total). Having identified the target variable as being *Buy*, we immediately carried out a train-test split of the training dataset to hold out 20 % of the originally available data. This way, in-house train (7988 rows) and validation (1997 rows) datasets were obtained, with only the former being used for model building, and the latter used to check for overfitting. Unless stated otherwise, future images were plotted using the data contained in the in-house train dataset.

Eleven new features were created (***Figure 5***). Based on our initial data exploration, we hypothesized that the time of year could be relevant to predict whether a purchase is made or not. As such, four new features were engineered from the original *Date* feature, namely the day of the week, month, meteorological season (all label-encoded), and a binary feature for whether access to the online store was made on a weekend or not. Thinking it likely that propensity to make a purchase could be related to whether the website was accessed via a computer or a mobile device, a *Device* feature was created from the available OS data. The remaining five features were created by combining information from two or more features into a single one. These include the average time spent on each of the visited FAQ, account management, and product pages. The total number of pages visited, regardless of type, and the average duration spent on each page was also added. All of these were created for the train and test sets provided, and for the in-house train and validation datasets. In the end, we were left with fifteen metric features and ten categorical features, and performed a coherence check of the data:

- We considered that an individual needs to visit at least one page to be regarded as a potential buyer, so the rows where no pages were visited were removed (4 and 6 in the train and train_in-house sets, respectively).
- We found it odd that people could visit pages on the website and yet have no time spent there. The number of observations where this happens is too large to consider removing them but does indicate a systematic issue. It could be that, in some cases, the time a user spends on a web page is not recorded.

During data exploration, we also noticed that some categories in the categorical features had very little representation and/or were not particularly informative (***Figure 6***). "Ubuntu", "Fedora", "Chrome OS", and "Other" categories from the *OS* feature were dropped as they represented a tiny percentage of the total accesses. The category "Other" from the *Type_of_Visitor* feature was dropped for the same reason. In total, we removed 1.14-1.16% of the train and the in-house train datasets (but not from the test datasets). In some cases, namely with the *Browser* and *Type_of_Traffic* features, we merged the underrepresented categories into an "Other" group, anticipating that their discriminatory benefit would not be high enough to offset the potentially detrimental impact of the increased dimensionality after transforming them into dummy variables. Finally, categorical features were label-encoded (***Figure 7***).

Afterwards we moved on to visualising the outliers in the metric features (***Figure 8***). We initially considered manually removing outliers guided by the Inter-Quartile Range (IQR) method, which would mean removing approximately 0.55% of the total number of observations in both the train and the in-house train datasets. However, acknowledging our limited understanding of the business domain and our inability to properly identify outliers, we decided against it. We also considered using DBSCAN as a multivariate outlier detection method but ultimately concluded that it was not improving our models (i.e, did not improve F1 score for in-house train or validation sets) and we were sacrificing potentially useful information for no apparent benefit.

Log transformation was applied to skewed metric features (***Figure 9***), resulting in remarkable levels of improvement of models that rely on normally-distributed data or linear relationships between features, such as Gaussian Naive Bayes and Logistic Regression [2]. For algorithms not relying on these characteristics to make good predictions, such as those based on Decision Trees, using the log-transformed or the original features provided similar results. As such, we decided to use the log-transformed metric features.

Given the multitude of features with different units and scales, the dataset was scaled before proceeding with feature selection. Because we decided not to remove any of the apparent outliers, we opted with RobustScaler, which removes the median and scales the data according to the specified quantile range [3]. After testing different ranges, we ended up using the 25-75 quantile range for data scaling.

Finally, we moved on to feature selection with the goal of removing redundant or irrelevant features. To decide whether to include or exclude the categorical features from our final model, we performed a Chi-squared test and a Mutual Information test. To select metric features, we looked at pairwise correlations using a heatmap of their Pearson and Spearman (***Figure 10***) correlations, and performed Recursive Feature Elimination (RFE), LASSO regression, RIDGE regression, and an ANOVA test (to check for variance). As with categorical features we also used Mutual Information criteria. For metric and non-metric features alike, we used Random Forest Classifier (with Gini and entropy) and AdaBoost as tools to determine the importance of each feature. We ended up selecting seven of the original twenty-five features (highlighted in ***Figure 11***) through a combination of the results reached with the methods mentioned above. Worth noting that *Season* was not selected because it is a binned, less-informative version of *Month*. Instead, *Type_of_Traffic* was selected because there was a noticeable difference between the proportion of returning and new visitors that end up making a purchase, and it was one of the few categorical features passing the Chi-squared test. From the metric features, *GoogleAnalytics_BounceRate* was discarded because of its high correlation (and therefore redundancy) with *GoogleAnalytics_ExitRate*.

To address the imbalance in the dataset and improve the predictions of our models we experimented with different resamplers, namely SMOTENC, RandomOverSampler, and RandomUnderSampler. We avoided resampling algorithms that are purely based on distances between points, such as SMOTE, TomekLinks, and SMOTETomek, since our dataset contained non-metric (and non-ordinal) features, for which distances are not representative of (dis)similarity. Even if we were to encode them as dummy variables, applying distance-based measures to a dataset containing both continuous and binary features is also not ideal [4].

4. Modelling

The main goal of this project was to improve the robustness of the predictions, maximizing the F1 score while preventing overfitting. As such, other considerations for choosing a modeling algorithm, such as efficiency, running time, and interpretability of the output, were mostly disregarded.

To guide our choice of a suitable model, we followed a top-down approach and started by performing a preliminary evaluation of various modeling algorithms. The goal was to have a general idea of which models would work best so that we would invest more time into these. We used a loop to run each model with its default hyper-parameters and visualized the F1 scores obtained for both the in-house train and validation sets. This allowed us to obtain a shorter list of models, for which we repeated the same process, this time also including three resampling algorithms (SMOTENC, Rando-

mUnderSampler, RandomOverSampler) with different sampling strategies to deal with data imbalance. This initial screening suggested *GradientBoosting*/*HistGradientBoosting*, *SupportVector*, and *Multi-layer Perceptron* Classifiers as producing the highest F1 scores, indicating that more complex algorithms are better suited to tackle this problem. Recognizing the limitations of this approach, we still tested and tweaked each individual model to understand if it was suitable for our application. Those deemed inadequate were discarded and are mentioned in the Supplementary Material. For each model, we had a few decisions to make, namely whether to use the "normal" or the log-transformed metric features, the labeled or the dummy-encoded categorical features, which resampling strategy to use (if any), and, if the model was performing relatively well, which hyperparameters to tune, guided by GridSearch or BayesSearch. Of note, whenever available, we attempted to tweak parameters designed to deal with imbalanced classification problems, such as class_weights in *Random Forest*. In some cases, we also attempted to change the probability threshold for what is considered to be 0 or a 1, but this approach had little success. The final results obtained for each modelling algorithm selected are shown in ***Figure 12***. Except for "F1 (val)", which was determined for the in-house validation dataset obtained before data preprocessing, the metrics displayed were obtained by performing cross-validation of the training set with *Stratified K Fold*.

4.1. Linear Models

A log transformation was applied to skewed features and greatly improved the performance of the linear models. The evaluated models consisted of the *Ridge Classifier*, and the *Logistic Regression*, with and without built-in cross-validation. Both had middle-of-the-pack performances, which is good considering they are simple models which achieved better results than some ensembles.

4.2. Discriminant Analysis

Both *Linear Discriminant Analysis* and *Quadratic Discriminant Analysis* had similar performances, as evaluated by the final metrics, with LDA favoring accuracy and precision and QDA displaying a higher recall. The slightly higher F1 score for LDA in the train set suggests the boundary between classes is closer to being linear than quadratic.

4.3. Support Vector Classification

Support Vector Machines are used to solve two-class pattern recognition problems [5], which it does by finding the hyperplane that differentiates the two classes the best [6, Chap. 7.4.7]. This is effective in high-dimensional space. In our case, the *Support vector machine* model had one of the best F1 scores for both train and validation datasets.

4.4. KNeighbors Classifier

KNeighbors Classifier also had high precision but very low recall and F1 scores. Like the previous algorithm, it is distance-based, assigning labels based on the similarity between observations, which is not the best strategy when working with datasets containing mixed data types [6, chap 5.4].

4.5. Naive Bayes

Gaussian Naive Bayes did not perform particularly well and was outperformed by *Complement Naive Bayes*. In fact, *Complement Naive Bayes* is designed to tackle some of the basic assumptions of *Naive Bayes* and is better at dealing with imbalanced datasets [7]. For Bayesian algorithms, resampling the data barely (if at all) improved model performance.

4.6. Decision Tree classifiers

As one of the more basic models, we tried the *Decision Tree Classifier* early on in our exploration, but quite quickly found its performance to be mediocre and discarded it entirely.

4.7. Ensemble methods

Ensemble methods differed greatly in their performance, as assessed by the final F1 scores of the training and validation datasets. *Bagging* ended up being one of the poorest performers, together with *AdaBoost* and *Extra Trees*, which were outperformed by simpler models like *Logistic Regression*. In stark contrast, *Gradient Boosting*, *Random Forest Classifier*, and *HistGradient Boosting* were our top performers, attaining the highest F1 scores for both the train and the validation datasets. *AdaBoost* is a specific instance of *Gradient Boosting* using exponential loss, which might be the factor that explains their difference in performance [8]. Also, while *Random Forest* chooses the optimum split criteria, *Extra Trees* chooses it randomly, adding an extra layer of randomization, to prevent overfitting [9]. However, the results obtained with *Random Forest* do not suggest that it has been overfitted.

4.8. Semi-supervised learning

In our preliminary screening, we tested *Label Propagation* and *Label Spreading*, which presented an average performance. However, these models are not the most indicated for our goal, and so were dropped and are further discussed in 1.7.4.

4.9. Multi-layer Perceptron

Multi-layer Perceptron(MLP) classifier was among the top 5 models when ranked by the final F1 score, which was expected, as even simple *MLPs* can represent complex functions. However, the model's performance did not change much when attempting to optimize hyperparameters, a process that proved more difficult and laborious than for most other modeling algorithms.

4.10. More ensembles

"*The whole is greater than the sum of its parts"* -Aristotle

4.10.1. Stacking

Stacked generalization (or Stacking) is an ensemble model that combines predictions from other machine learning models, as these can sometimes outperform even the best performing of the used algorithms [10]. The base models used were *Random Forest Classifier*, *Histogram-based Gradient Boost-*

ing Classifier, and *SVC*, as these were models which had previously performed well by themselves. We tested both *Gradient Boosting* and *HistGradient Boosting* and ended up selecting *HistGradient Boosting* based on trial and error. We did not use both as their algorithms are not that different. For our choice of the meta-model, we tried both *Logistic Regression* and *Ridge Regression*, but found that this did not improve the predictions made using the base models separately.

4.10.2. Voting

The *Voting Classifier* combines the predictions from different models and uses either the majority (hard voting) or the average probabilities (soft voting) to make its own predictions [10]. Just as we did for *Stacking*, we used *Random Forest Classifier*, *Histogram-based Gradient Boosting Classifier*, and *SVC* due to their good solo performances. Both hard and soft voting were able to reach better F1 scores than any of the models used, but it was the former that achieved the best results.

4.11. Conclusion

After testing each of these models individually, we decided, based on our understanding of the models and their performance, that the most promising models were *HistGradientBoosting*, *GradientBoosting* and *RandomForest*. In addition, stacking and voting seemed promising as well.

4.12. Hyper-parameter optimisation

Optimizing hyper-parameters is the most computationally expensive part of our process. We, therefore, had to structure much of our thought process around this. Our methodology focused on testing and eliminating models based on lower performance by running simpler tests. This was done in an attempt to maximize time spent running hyper-parameter optimization versus the model performance.

The hyper-parameters were optimized initially using *GridSearchCV* and *RandomSearchCV*, but we later went on to use *BayesSearchCV* [11], as it should outperform both the aforementioned methods. It does this by using *Bayesian* optimization, which allows it to avoid testing all combinations such as *GridSearch* and improves on *RandomSearch* by taking advantage of the structure of the search space.

That gave us close to optimal results for each of the models by sacrificing computational power and time. During hyper-parameter search, we also retested each resampler to ensure optimal performance. During this optimization, the model that performed the best was *RandomForestClassifier*.

5. Conclusion

Through an extensive analysis of the data and plausible models to use to maximize our performance, the final choice for this project was *Voting* (hard) using three of our best performing models, *RandomForestClassifier*, *HistGradientBoostingClassifier* and *SVC*. Their hyperparameters can be seen in ***Listing 1***. The model produced the same f1 score on both the dataset with and without outliers, which is promising in regards to the robustness to outliers. The model had very similar scores in both in-house testing and on Kaggle, and it showed no apparent signs of either overfitting or underfitting. We are confident that the model will continue to perform on the future data without any significant fall-off.

With this, we believe *TechScape* can work on capturing more sales through more aggressive marketing where our model predicts it to be fitting. For startups, it is important to stay lean when it

comes to expenses by making each dollar create as much value as possible. Robust prediction models allow this by consistently narrowing down the aim for marketing campaigns while also identifying the customers that are more likely to interact positively with such campaigns.

5.0.1. Improvements

Through substantial work on this data set, we noticed some gaps that we believe could be valuable assets for *TechScape*. The prediction rates of our models can be considerably improved if more data is gathered. We, therefore, have some suggestions: storing a variable on recency of purchase, what page users leave the website from (crucial in identifying usability pain points), dates of campaigns (easier to identify successful campaigns), storing profit from purchase, making it a regression problem rather than a classification problem and specific product pages visited.

5.0.2. Deploying the model

Using the model correctly will also be an important aspect of creating as much value as possible. We suggest using it in batch prediction, where every week/month predictions can be made based on new or recurring customers and their behavior. We feel this is more fitting as startups often deal with smaller customers bases and it is probably not necessary and might be too expensive to do predictions continuously.

Bibliography

[1] Stratos Idreos, Olga Papaemmanouil, Surajit Chaudhur, "Overview of data exploration techniques,"

[2] Changyong FENG and Hongyue WANG and Naiji LU and Tian CHEN and Hua HE and Ying LU and Xin M. TU, "Log-transformation and its implications for data analysis," *Shanghai Arch Psychiatry*, vol. 26, pp. 105–109, 2014. DOI: 10.3969/j.issn.1002-0829.2014.02.009.

[3] "Sklearn.preprocessing.robustscaler," Sklearn. (), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html (visited on 12/26/2021).

[4] D. Randall Wilson,Tony R. Martinez, "Improved heterogeneous distance functions," *Journal of Artificial Intelligence Research*, vol. 6, pp. 1–34, 1997.

[5] M.-S. Mushtaq. "Methodologies for subjective video streaming qoe assessment." (2017).

[6] A. D. John D. Kelleher Brian Mac Namee, *Fundamentals of Machine Learning for Predictive Data Analytics*. Cambridge MA: The MIT Press, 2015.

[7] Jason D. M. Rennie, Lawrence Shih, Jaime Teevan, David R. Karger, "Tackling the poor assumptions of naive bayes text classifiers," 2003.

[8] "Gradient boosting classifier," Sklearn. (), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html#sklearn.ensemble.GradientBoostingClassifier (visited on 12/23/2021).

[9] L. W. Pierre Geurts Damien Ernst, "Extremely randomized trees," *Machine Learning*, vol. 3, no. 42, pp. 1–40, 2006. DOI: 10.1007/s10994-006-6226-1.

[10] "Ensemble methods," Sklearn. (), [Online]. Available: https://scikit-learn.org/stable/modules/ensemble.html (visited on 12/23/2021).

[11] G. L. Iaroslav Shcherbatyi Tim Head. "Bayes search cv." (2017), [Online]. Available: https://scikit-optimize.github.io/stable/auto_examples/sklearn-gridsearchcv-replacement.html.

[12] "Bayesian gridsearchcv," Sklearn. (), [Online]. Available: https://scikit-optimize.github.io/stable/auto_examples/sklearn-gridsearchcv-replacement.html (visited on 12/26/2021).

[13] "Compare the effect of different scalers on data with outliers," Sklearn. (), [Online]. Available: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html (visited on 12/26/2021).

[14] "Smotenc," Imbalanced-learn. (), [Online]. Available: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTENC.html (visited on 12/23/2021).

[15] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority oversampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[16] D. Pugh. “Balancing datasets and generating synthetic data with smote,” DATA SCIENCE CAMPUS, ONS. (07/18/2019), [Online]. Available: https://datasciencecampus.github.io/balancing-data-with-smote/ (visited on 12/26/2021).

[17] “Smotetomek,” Imbalanced-learn. (), [Online]. Available: https://imbalanced-learn.org/dev/references/generated/imblearn.combine.SMOTETomek.html (visited on 12/26/2021).

[18] “Combination of over- and under-sampling,” Imbalanced-learn. (), [Online]. Available: https://imbalanced-learn.org/dev/combine.html (visited on 12/26/2021).

[19] “Random over sampler,” Imbalanced-learn. (), [Online]. Available: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html (visited on 12/23/2021).

[20] “Random under sampler,” Imbalanced-learn. (), [Online]. Available: https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html (visited on 12/23/2021).

[21] “Tomeklinks,” Imbalanced-learn. (), [Online]. Available: https://imbalanced-learn.org/dev/references/generated/imblearn.under_sampling.TomekLinks.html (visited on 12/23/2021).

[22] “Under-sampling,” Imbalanced-learn. (), [Online]. Available: https://imbalanced-learn.org/dev/under_sampling.html (visited on 12/26/2021).

[23] “Anova,” Sklearn. (), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html (visited on 12/26/2021).

[24] D. N. Reshef, Y. A. Reshef, H. K. Finucane, *et al.*, “Detecting novel associations in large data sets,” *Science*, vol. 334, no. 6062, pp. 1518–1524, 2011. DOI: 10.1126/science.1205438. eprint: https://www.science.org/doi/pdf/10.1126/science.1205438.

[25] J. F. Trevor Hastie Robert Tibshirani, *The elements of statistical learning*. Springer.

[26] Sklearn, *Sklearn naive bayes bernoullinb*, Available at https://scikit-learn.org/stable/modules/naive_bayes.html#bernoulli-naive-bayes (2021/12/22).

[27] “Nearest centroid,” Sklearn. (), [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html#nearest-centroid-classifier (visited on 12/26/2021).

[28] “Label spreading,” Sklearn. (), [Online]. Available: https://scikit-learn.org/stable/modules/semi_supervised.html#label-propagation (visited on 12/23/2021).

[29] Xiaojin Zhu, Zoubin Ghahramani, “Learning from labeled and unlabeled data with label propagation.,” 2002.

[30] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[31] T. Yiu. “Ensemble methods,” towardsdatascience. (07/12/2019), [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2 (visited on 12/24/2021).

[32] “Voting classifier,” Sklearn. (), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html (visited on 12/23/2021).

List of Figures

1. Initial Features 17
2. Histograms of metric features in the train dataset originally provided. 17
3. Initial exploration of the categorical features present in the original train dataset. 18
4. Relation between the number of accesses to the online store and the number of new COVID-19 cases in Portugal. 18
5. Engineered Features 19
6. Categorical features in the in-house train dataset before preprocessing. The in-house train dataset was obtained by performing a train-test-split of the originally provided train dataset. 19
7. Categorical features in the in-house train dataset after preprocessing. The in-house train dataset was obtained by performing a train-test-split of the originally provided train dataset. 20
8. Boxplots of the metric features in the in-house train dataset. No outliers were removed before modelling. Instead, data was scaled using RobustScaler. 20
9. Histograms of the metric features in the in-house train dataset after log-transformation of skewed features. 21
10. Correlation heatmaps using both Pearson and Spearman coefficients to detect linear and monotonic relationships (respectively) between pairs of features. 21
11. Features selected (blue/green) for modelling. Only 7 of the initial 25 features (original plus engineered) were selected based on the feature selection methodologies employed. GoogleAnalytics_BounceRate was discarded due to its high correlation with GoogleAnalytics_ExitRate. Season was discarded as it is simply a binned version of Month. 22
12. Final metrics obtained for each modelling algorithm. Algorithms are sorted by the F1 score obtained for the in-house train dataset by cross-validation. 22

List of Listings

1. Voting model 23

Appendix

1. Supplemental

1.1. Hyper-parameter optimization

A big part of our modeling process is the use of hyper-parameter optimization. To be able to perform as extensive optimizations as possible this step needed to be optimized and for this we ended up using both *GridSearchCV* and *BayesianSearchCV*. The latter uses *Bayesian* optimization to find extrema, which is done by approximating functions with other functions, known as surrogate functions, and, based on these functions, promising areas can be identified. These areas are then tested again, and, after multiple iterations, the global extremes are mostly found. This allows *BayesianSearch* to be more efficient than *GridSearch* [12].

1.2. Pipeline

We made use of imblearn pipelines to sequentially apply sampling and a estimator while performing cross validation.

1.3. Scaling Methods

Standard Scaler removes the mean and scales the data to unit variance. This scaler cannot guarantee balanced feature scales in the presence of outliers [13]. Unlike it, Robust Scaler is based on percentiles and, therefore, is not influenced by outliers [13]. This Scaler removes the median and scales the data according to the quantile range [3].

1.4. Sampling Methods

In order to deal with the imbalance in our data, we used sampling methods. One of those methods was SMOTENC, a variant of SMOTE, specifically designed for datasets containing both numerical and categorical features [14]. SMOTE stands for Synthetic Minority Oversampling Technique and consists of an oversampling method to solve the imbalance. This method oversamples the minority class by creating synthetic examples rather than by oversampling with replacement [15]. The synthetic instances are generated according to the neighborhood of each example of the minority class [16]. Another variant of SMOTE is SMOTE Tomek, which is a hybrid technique that combines both undersampling and oversampling techniques to optimize the performance of classifier models for the samples created. It oversamples using SMOTE and cleans overlapping data points using Tomek links [17] [18]. As explained above, we ended up not using this method.

Another sampler used was RandomOverSampler, which, as the name suggests, over-samples the minority class by randomly picking samples with replacement [19], while RandomUnderSampler, also used, under-samples the majority-class by randomly picking samples with and without replacement [20].

Another technique that we considered that was not covered in the syllabus was *TomekLinks*. *TomekLinks* is a neighbor-based approach to undersampling that works by checking which observations from one class have closest neighbors from the other class and vice versa. Then, the majority class observation is removed. Removing observations like this makes for a clearer divide between the classes [21] [22].

1.5. ANova

ANova or *Analysis of Variance* is a test that is used for comparing means of more than two groups. ANOVA can be either one-way or two-way. In our case, we used one-way ANOVA, since we only had one independent feature. ANOVA uses an f-test to check the equality of the means of the groups. [23]

1.6. Maximal Information Coefficient

"Maximal information Coefficient is an information theory-based measure of association that can capture a wide range of functional and non-functional relationships between variables." [24]

Maximal Information Coefficient is used for identifying associations between features in hopes of finding the best set of features. MIC equals the percent of feature one that can be explained with feature two. It takes values from 0 to 1, and the closer the value to 1, the higher the statistical dependence. Advantages of MIC are that it is easy to understand and interpret, that it's very robust to outliers and that it works on a multitude of different linear and non-linear relationships. The biggest disadvantage is the sheer computational power needed to efficiently use it. [24]

1.7. Models

1.7.1. Discriminant analysis

Discriminant analysis assumes that the data used comes from a gaussian distribution, the Bayesian rule is then used to classify the data using the highest likelihood. Discriminant functions are used to find how probable it is for a data point to be in a class. We used two types of discriminant analysis where linear occurs when we assume equal covariance among the classes- Without this assumption we have the quadratic kind [25].

1.7.2. BernoulliNB

BernoulliNB was discarded early on in our testing, as it assumes that the data is distributed according to multivariate Bernoulli distributions, which is not the case in our data [26].

1.7.3. Nearest centroids

Nearest centroids is a simple algorithm that creates centroids for each class present in the training data by using the means, and then assigns the classes in new data based on the proximity to the centroids [27].

1.7.4. Label Spreading and Label Propagation

The two algorithms behave similarly, but *Label spreading* uses an affinity matrix on the normalized graph *Laplacian* and soft clamping across the labels. This different affinity matrix makes label spreading more robust to noise, while the use of clamping allows it to change the weight of the original label distribution based on a clamping factor [28].

To go into the inner workings of the algorithms, they are known as semi-supervised learners, which means that a subset of the data is labeled while the rest is not. Our application is probably not the

intended use, but it seems to perform well nonetheless. The algorithms are based on creating graphs with edges between similar nodes. These graphs are then randomly walked from each unlabeled node, where each walk ends when it encounters a labeled node, which is therefore known as absorbing states. The class of the unlabeled node is then decided based on what absorbing state class the majority of the random walks end at.

This concept can be reformulated using matrixes, wherein the label propagation matrix represents the probabilities of going from one node to another, which means a similarity matrix in the case of *label propagation* and a normalized *Laplacian* in the case of label spreading. If we then multiply this matrix by itself enough times, it will stabilize giving us the final probabilities for each of the classes for each of the nodes. The highest probability for each node is then selected to assign a class [29].

1.8. Voting

Voting is classified as an ensemble model, so it is important to understand what *ensembles* are before delving into *Voting*.

"Ensemble learning is a learning paradigm that, instead of trying to learn one super-accurate the model focuses on training a large number of low-accuracy models and then combining the predictions given by those weak models to obtain a high-accuracy meta-model." [30]

Ensembles work on the *wisdom of the crowd's* concept, which is the idea that a larger number of models will on average outperform the individual models. [31] The upside and the reason for using voting is to get multiple average performing, most often weak learners that are fast but not that precise, to fill in for each other's shortcomings to make a model that would ideally outperform any of the individual models used in the ensemble.

When handling classification problems, voting has two approaches: *hard voting* and *soft voting*. Hard voting chooses the prediction with the *highest number of votes*, while soft voting chooses the prediction with the *highest total probability* by combining probabilities from all the models.

Since voting works based on combining multiple machine learning models, all the models must have similar efficiency, otherwise one model will be dragging down the best-performing ones, making it so that a simpler model could outperform voting in the end. Also, it is important to take notice of your dataset and the problem, because it is not assured that the more complex model will outperform simpler models. An example of that is when using linear regression on a simple dataset with high linearity will net much better results than the combination of more complex models using voting [32]. As the complexity of the model increases, so does the necessary computational power as well.

1.8.1. Voting Model Performance

When we started with using *VotingClassifier*, our thought process was to use previous best performing models to try to get something out of each of them to improve the final score. So models used were *SVC*, *RandomForestClassifier* and *HistGradientBoosting*, as can be seen in ***Listing 1***.

The first classifier used was *SVC* with probability = True, which means that probability estimates will be calculated using five-fold cross-validation, which will slow down the model but allowed the results to also be used to test for soft voting. We kept the C=1 default value for the regularization parameter as lower values did not improve the score and higher values would increase the likelihood of overfitting. (***Listing 1 line 4***).

The second classifier used was *RandomForestClassifier* with n_estimators=250, which is an

increased number of trees from default 100 to 250, criterion='entropy' is used for information gain in favor of Gini. Gini does only basic operations, which makes it faster, but it performed worse than entropy, which is calculated using logarithms that make it computationally slower. Max_depth =12, min_samples_leaf=5 are the maximum depth of a tree and a minimum number of samples that have to be on each leaf respectively. The best values for both parameters were found using BayesianSearchCV. (***Listing 1 line 5***)

The third and last classifier used was *HistGradientBoostingClassifier* with learning_rate=0.1 which is its default value. Lower learning_rate improves the model but greatly increases the computational power necessary. Loss=auto is also its default value, which chooses from binary_crossentropy / categorical_crossentropy depending on the problem at hand. Categorical_features = categorical_indexes, which is an integer array of indices indicating categorical features. (***Listing 1 line 6***)

Once all models were trained, they were used with *VotingClassifier*, that had its voting parameter set to 'hard', meaning the majority vote wins. The final f1 score achieved was 0.73333, with no apparent signs of either underfitting or overfitting, and was deemed satisfactory by us. (***Listing 1***)

2. Figures

Feature	Data type	Description
AccessID	Int64	Unique identification of the user access to the website
Date	Object	Website visit date
AccountMng_Pages	Int64	Number of pages visited by the user about account management
AccountMng_Duration	Float64	Total amount of time (seconds) spent by the user on account management related pages
FAQ_Pages	Int64	Number of pages visited by the user about FAQ, shipping information and company related pages
FAQ_Duration	Float64	Total amount of time (seconds) spent by the user on FAQ pages
Product_Pages	Int64	Number of pages visited by the user about products and services offered by the company
Product_Duration	Float64	Total amount in time (seconds) spent by the user on products and services related pages
GoogleAnalytics_BounceRate	Float64	Average bounce rate value of the pages visited by the user, provided by google analytics
GoogleAnalytics_ExitRate	Float64	Average exit rate value of the pages visited by the user, provided by google analytics
GoogleAnalytics_PageValue	Float64	Average page value of the pages visited by the user, provided by google analytics
OS	Object	Operating System of the user
Browser	Int64	Browser used to access the webpage
Country	Object	The country of the user
Type_of_Traffic	Int64	Traffic Source by which the user has accessed the website (e.g., email, banner, direct)
Type_of_Visitor	Object	User type as "New access", "Returner" or "Other"

Figure 1: Initial Features

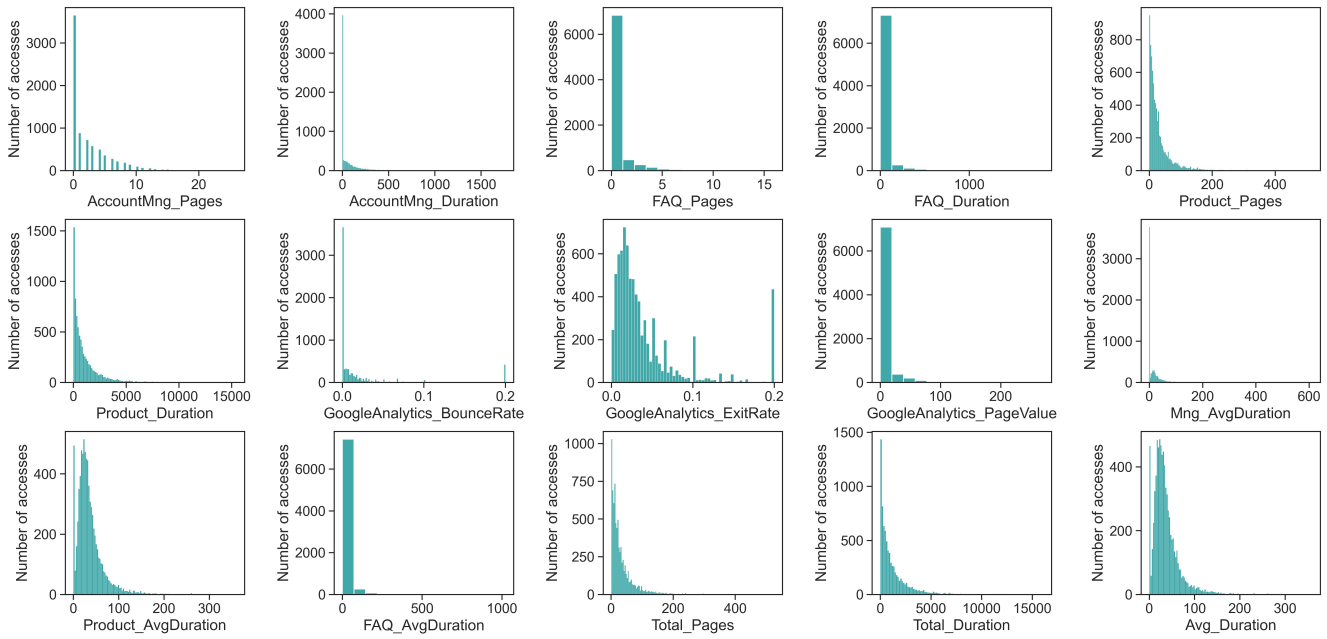

Figure 2: Histograms of metric features in the train dataset originally provided.

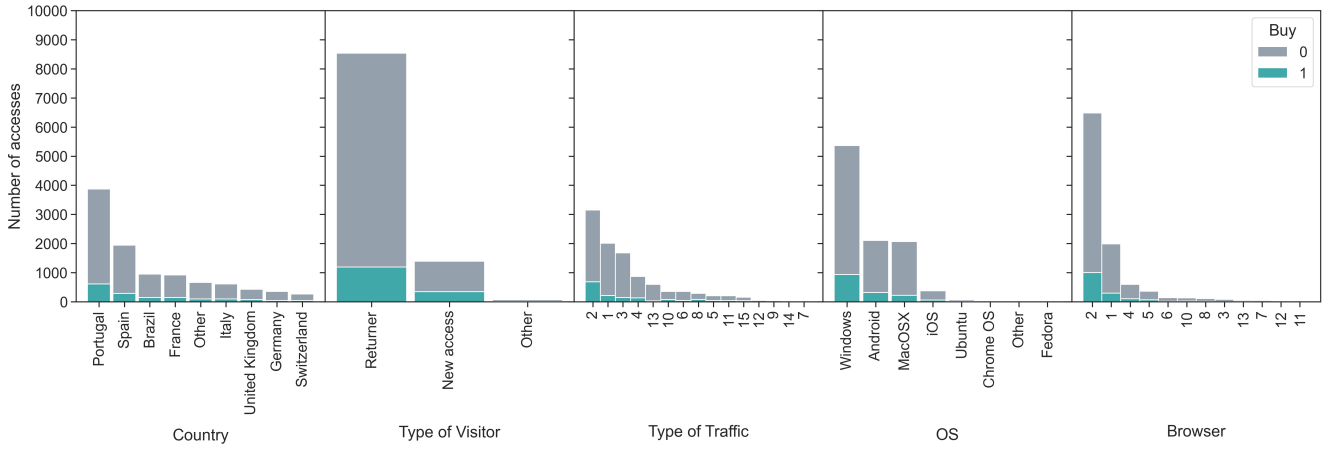

Figure 3: Initial exploration of the categorical features present in the original train dataset.

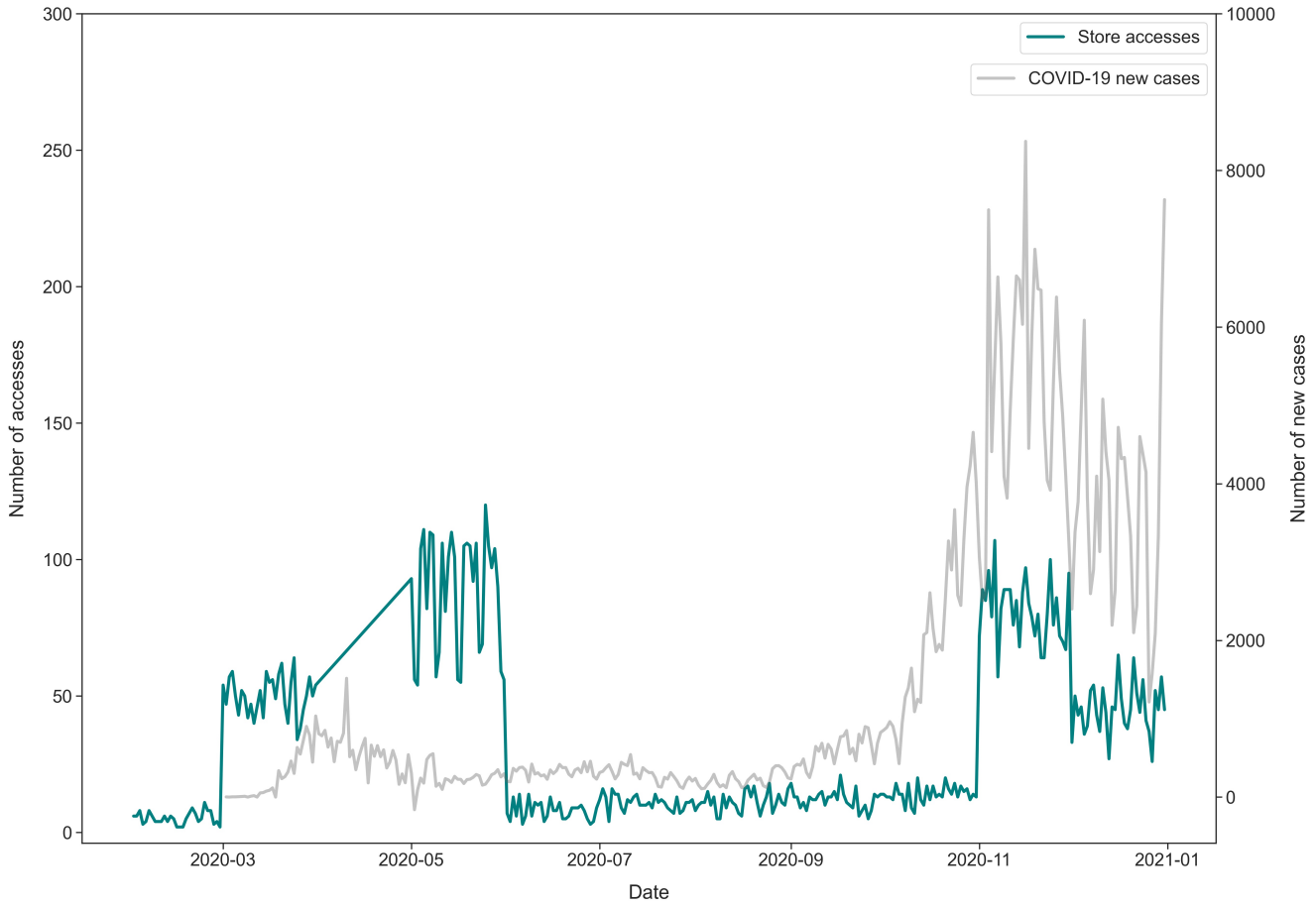

Figure 4: Relation between the number of accesses to the online store and the number of new COVID-19 cases in Portugal.

Feature	Description
Week_Day	Labeled category; day of the week when the website was accessed (0-6, 0 is Monday)
Weekend	Binary; indicates if the access occurred on a weekend (1) or not (0)
Month	Labeled category; month when the website was accessed (1-12)
Season	Labeled category; meteorological season when the website was accessed (1-4, 1 is Spring)
Device	Labeled category; device used to access the website (1 Desktop, 2 Mobile)
Mng_AvgDuration	Float; average time (seconds) spent on management pages
FAQ_AvgDuration	Float; average time (seconds) spent on FAQ, shipping information, and company related pages
Product_AvgDuration	Float; average time (seconds) spent on product and service related pages
Total_Pages	Integer; total number of pages (Management, FAQ, Product) visited per website access
Total_Duration	Float; total amount of time (seconds) spent by the user on all accessed pages
Avg_Duration	Float; average time (seconds) spent on the pages (Management, FAQ, Product) accessed

Figure 5: Engineered Features

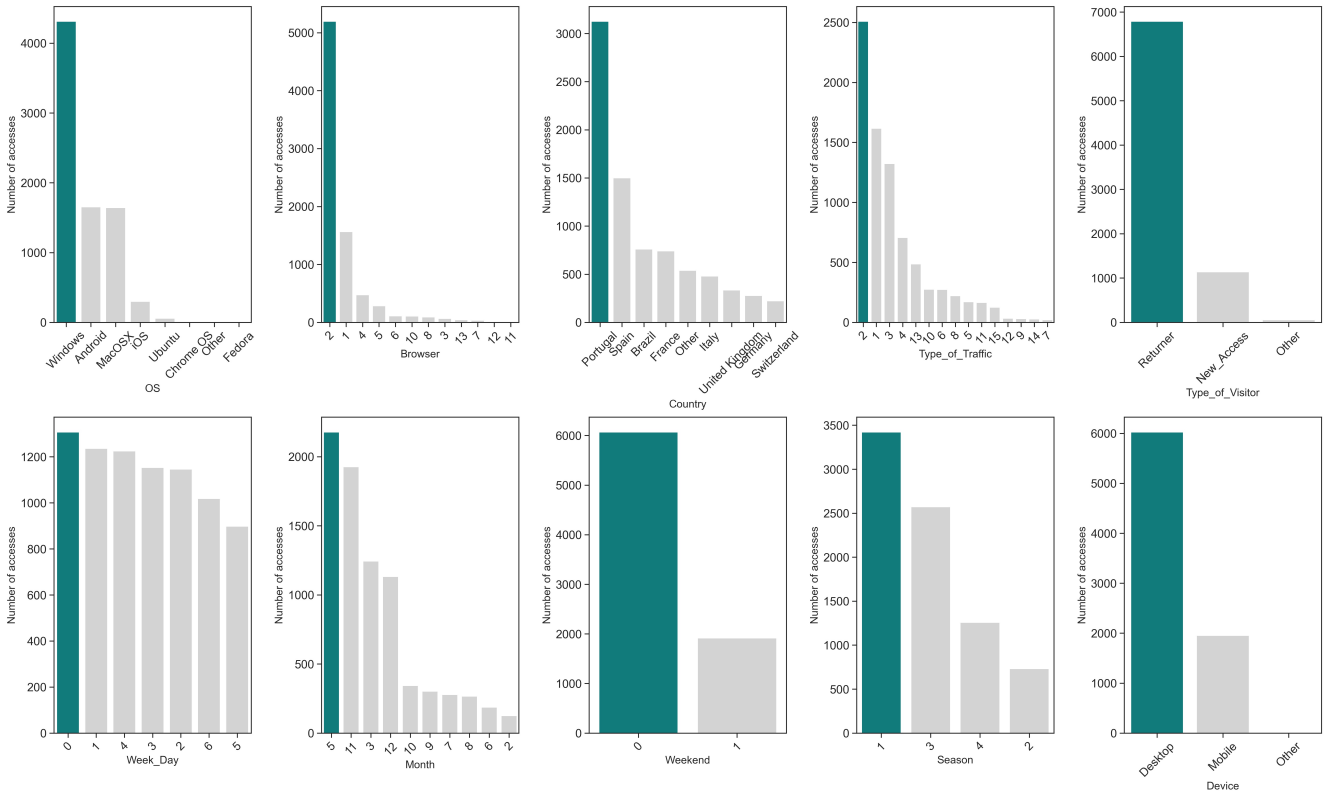

Figure 6: Categorical features in the in-house train dataset before preprocessing. The in-house train dataset was obtained by performing a train-test-split of the originally provided train dataset.

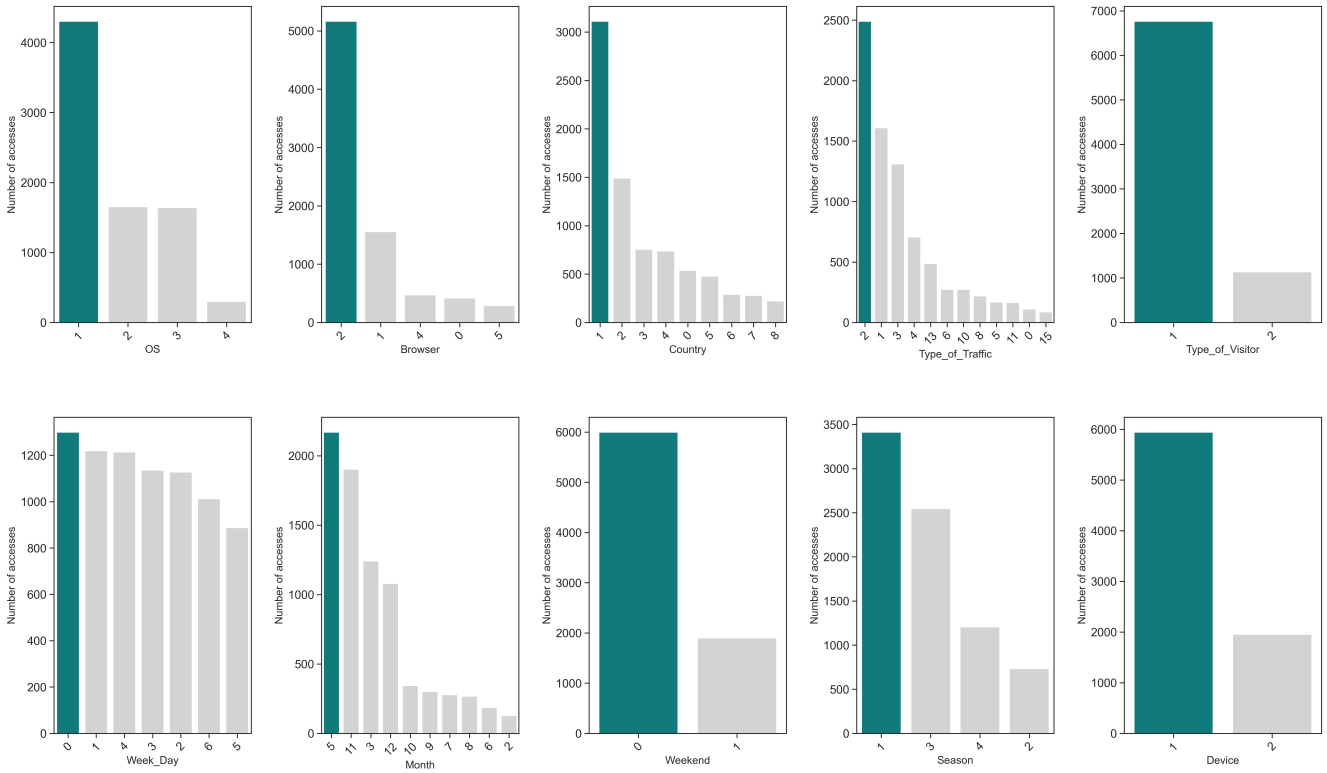

Figure 7: Categorical features in the in-house train dataset after preprocessing. The in-house train dataset was obtained by performing a train-test-split of the originally provided train dataset.

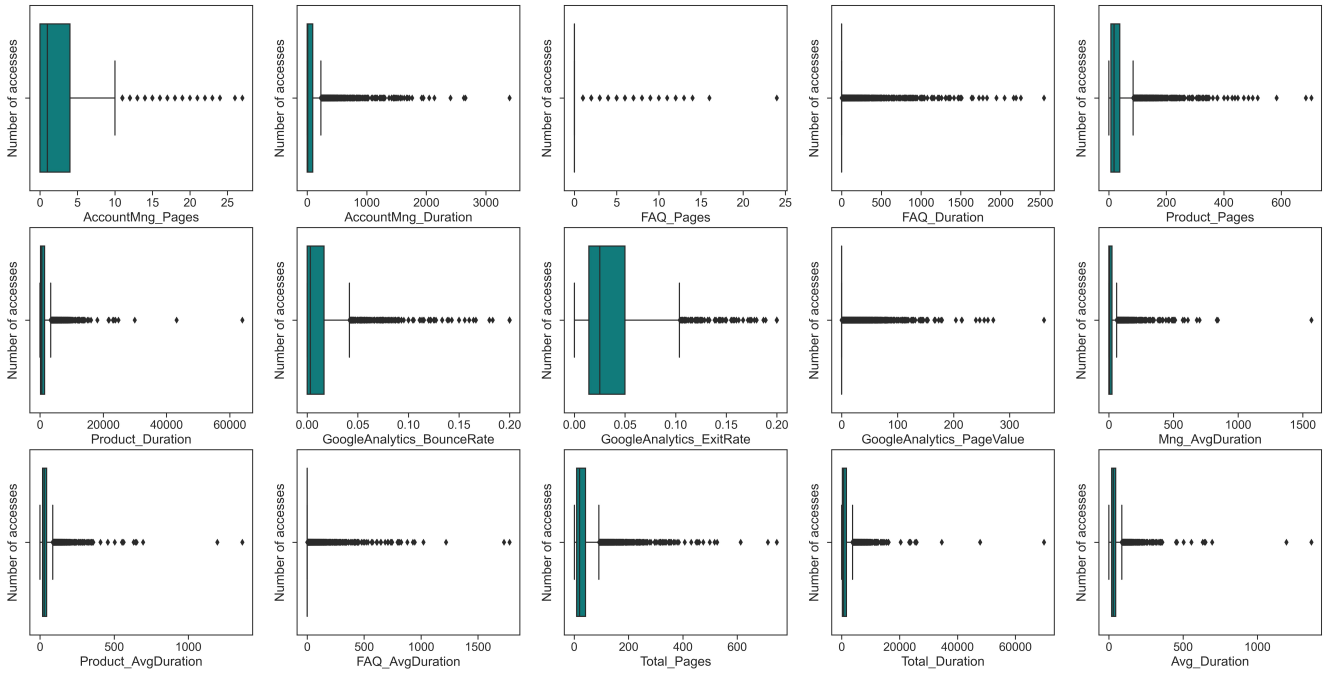

Figure 8: Boxplots of the metric features in the in-house train dataset. No outliers were removed before modelling. Instead, data was scaled using RobustScaler.

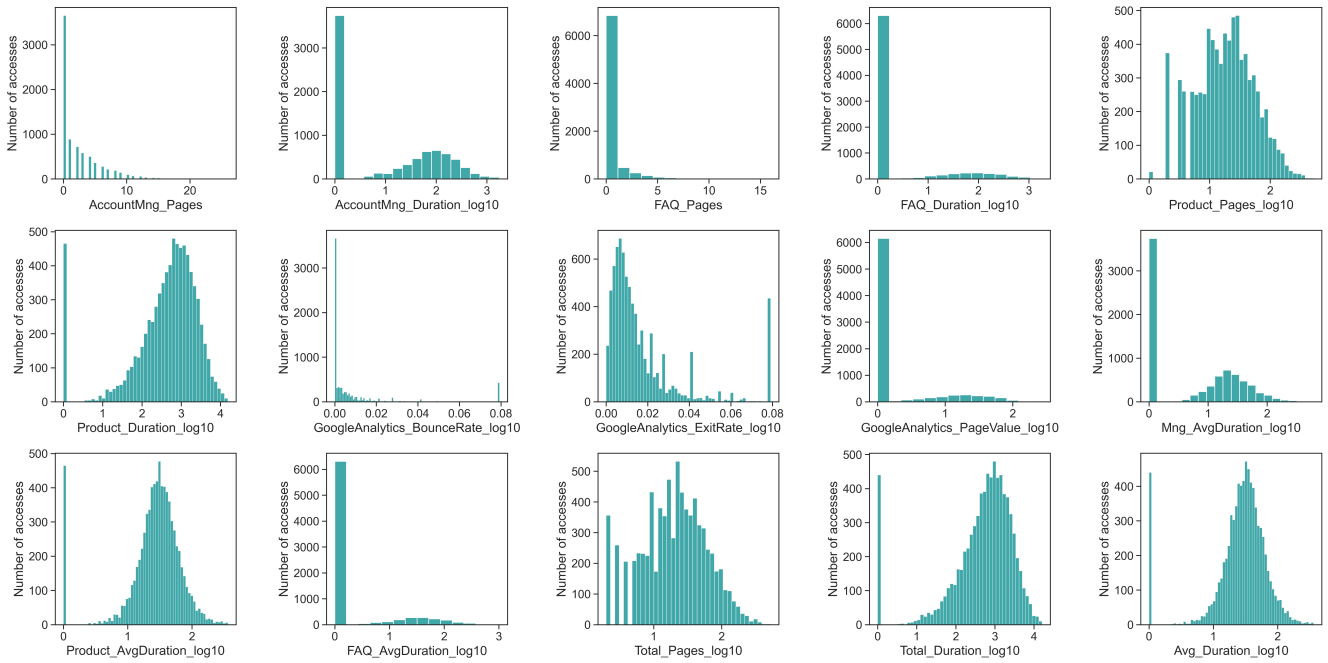

Figure 9: Histograms of the metric features in the in-house train dataset after log-transformation of skewed features.

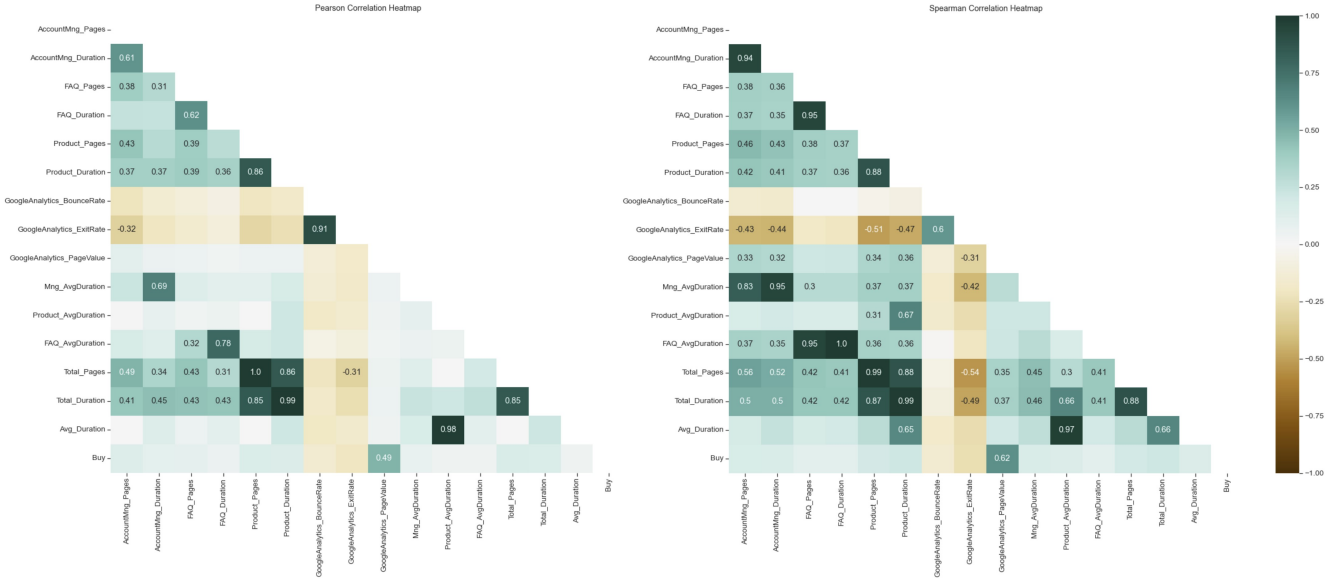

Figure 10: Correlation heatmaps using both Pearson and Spearman coefficients to detect linear and monotonic relationships (respectively) between pairs of features.

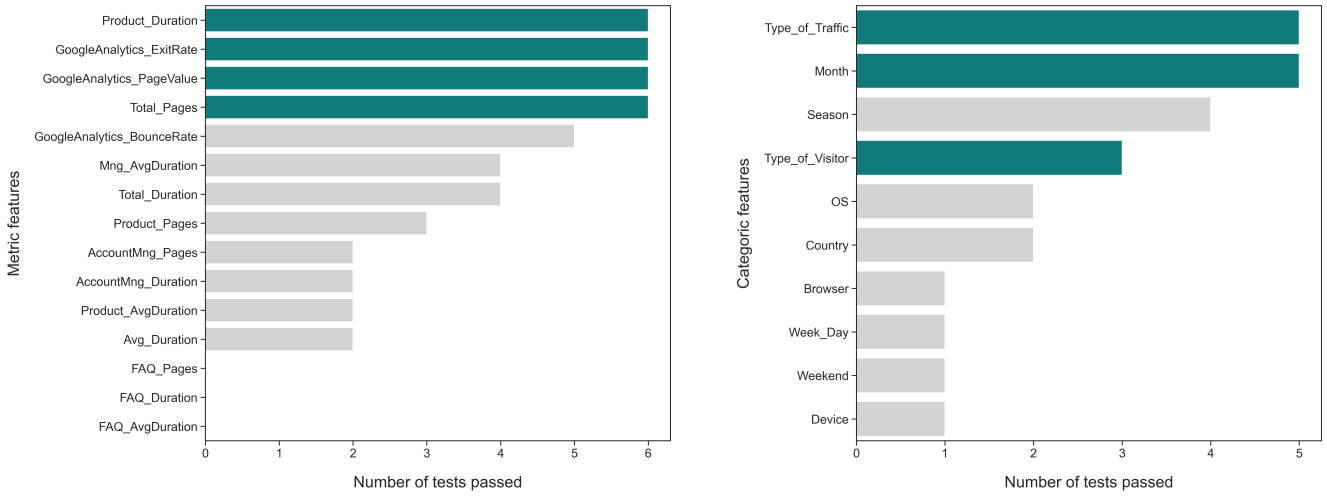

Figure 11: Features selected (blue/green) for modelling. Only 7 of the initial 25 features (original plus engineered) were selected based on the feature selection methodologies employed. GoogleAnalytics_BounceRate was discarded due to its high correlation with GoogleAnalytics_ExitRate. Season was discarded as it is simply a binned version of Month.

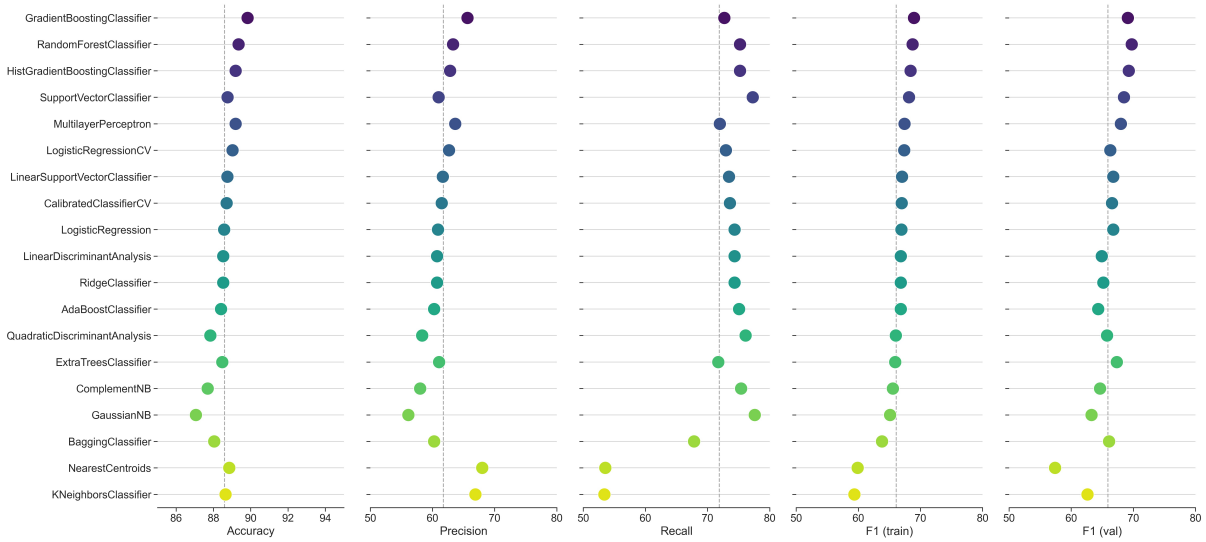

Figure 12: Final metrics obtained for each modelling algorithm. Algorithms are sorted by the F1 score obtained for the in-house train dataset by cross-validation.

3. Listings

3.1. Listing 1.

Listing 1: Voting model

```
submissionfeatures = metric_features_log+ohc_df_feats
resampler = RandomOverSampler(sampling_strategy = 0.50, random_state = 123)

clf1 = SVC(probability = True, C = 1, random_state = 123)
clf2 = RandomForestClassifier(n_estimators = 200, criterion = "gini", max_depth = 20,
 min_samples_leaf = 5, random_state = 123)
clf3 = HistGradientBoostingClassifier(learning_rate = 0.1, loss = "auto", categorical_features
  = categorical_indexes, random_state = 123)

model = VotingClassifier(estimators = [("svc", clf1), ("rf", clf2), ("hgb", clf3)], voting = "
 hard")

data_submission = data_ohc_scaled[features]
test_data_submission = test_ohc_scaled[features]
```