Machine Learning Project

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Group 13

Dafnides, Nicholas: 20210648

Faisca Guedes, Rodrigo Miguel: 20210587

Hadidi Pereira, Miriam Natasha, 20210644

Langenstein, Amelie Florentine: 20210637

Rensink, Roeland: 20200759

December, 2021

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

Table of Contents

1. Introduction & Hypothesis

The Portuguese startup *TechScape* wants to address the need to detach from the digital world and sustain mental health, which can be difficult in today's times. They developed digital detox products first launched at their online shop in the beginning of 2020. After financial difficulties in March 2020, the company was out of function in April 2020, but restarted their activities in May 2020. To increase their sales, *TechScape* hired us to analyze the online actions of their customers and predict customers that are more likely to buy their products.

The object of this project is to create a model based on a two-class classification problem to predict the target variable for a dataset the model has not seen.

To test the performance of our model we split the training data beforehand into a smaller training dataset with 70/80% of the records and a validation dataset with 20/30% of the records. The training dataset will be used to create the predictive model and the validation dataset will be used to evaluate the performance of the model. This prevents the model from overfitting and helps to build an optimal predictor that can be applied on a new dataset.

When planning out our strategy, we hypothesized that a random forest method would be the best model for the techscape problem, due to the highly categorical nature of our data. Decision trees are best used in classification problems with a discrete target, and do not work as well when considering many classes, so we agreed that this would be the ideal model in complex learning methods such as ensemble.

2. Data Exploration and Pre-Processing

We briefly explain here the general exploration and preprocessing of the data and will explore how each model handled data types more specifically in its respective section if they stray from the norm.

The dataset from Techscape | Ecommerce contains 9999 records. Each record consists of 17 attributes, 12 numerical and 5 categorical variables. All of them have no missing values. The target variable *Buy* is a binary variable indicating whether a customer completed his/her purchase or not. Through analysis, we have found that the dataset is imbalanced. Only 16% of the records indicate a purchase (Buy = 1), while the other 84% indicate unrealized purchases (Buy = 0).

In the pandas dataframe resulting from the read_csv function, some features did not have the correct data type or needed to be transformed to facilitate further analysis. For example, the date column was in the form of the data type object. It needed to be transformed to either datetime, a numerical version of datetime depending on the model, or to an integer that represented the month for purposes of analysis. If a model could not parse the actual datetime value, we used a numerical version of the datetime so that we could still make use of the unique values.

The Access ID column only contains unique values but does not provide additional information that could be analysed. Therefore, it was set as the index of the data frame.

Date – As an initial approach, we split the date by months because this can usually be revealing of trends in customers interest in our products. There is also interest treating it as an ordinal value computationally. From Appendix A1 we can observe that the months where our business had a higher access count were May (first month after restarting activities), November and December (months where the weather shift is more noticeable and people stay most of their time indoors, enhancing the need for digital detox) and March (where the first lockdown happened).

OS and Browser – Typically, most users of the same operating system utilize the same collection of browsers. While trying to find a correlation between OS and Browser chosen, we decided first to look at the totals for each attribute (figures A2 and A3). As suspected, most of the OS and Browser records were distributed in a small subset of categories. OS wise, our customers are represented by 'Windows', 'Android', 'MacOSX' and 'iOS'. When we try to relate the OS used with the Browser used, we can clearly observe in figure A4 that there's a significant correlation between them - apart from iOS where you could argue between browser 1 and 2 but this is the smallest subset of our OS records. Additionally, when we try to correlate these attributes with the target variable individually, there's no clear suggestion of an existing direct correlation as shown in figures A5 and A6. From these observations the decision can be made to drop the attribute 'Browsers' and to remove the records that had the OS different from the four major categories in order to develop the model. In the model predictions, if the OS is not one of these four, the entry on the table is modified to Windows as this is the great representative category of the attribute.

Country – Although most of the accesses are from Portugal, there isn't a clear demographic separation to where we might consider dropping/replacing records on rows based on this specific attribute. Relating it to the target variable, there isn't a clear suggestion of a correlation, so the choice to drop this attribute can be made (see figure A10).

Type of Traffic and Type of Visitor – These attributes are very similar in the sense where they both show relevant correlation regarding the target variable, and they show multiple categories of interest (see figures A7 and A8).

Numerical Data – When looking at the correlation matrix of figure A9 there's an obvious correlation between AccountMng_Pages and AccountMng_Duration (0.9), between FAQ_Pages and FAQ_Duration(1) and between Product_Pages and Product_Duration (0.9). This suggests that for performance improvement/noise reduction these variables could be dropped. It's worth mentioning that the GoogleAnalytics_PageValue has a high correlation with the target variable Buy (0.6), which can be a hint to explore the importance of this attribute in the different models.

In early models, eliminating certain variables with high p-values did not improve the validation or test scores. Because of this, we kept all variables to begin with and explored how each model would perform with default parameters. However, we learned that in certain models, such as random forests, the nature of the model means that not all variables were necessarily chosen by the algorithm to be used in each branch - and possibly never chosen at all. Random forests tended to perform better than the basic learners. In addition, we understood that this may be a superficial effect and that certain variables should be eliminated as a result of their internal correlation to each other. For example, operating systems and internet browsers were highly correlated. Thus, it did not make sense to include both features and could result in overfitting when including irrelevant variables.

Some models are exclusively distance based and categorical attributes are encoded into numerical dummy values for posterior normalization. However, in some models like decision trees, some categorical attributes like 'OS', 'Country' and 'Type of Visitor' whose parameters are unrelated in a distance evaluation parameterization would benefit from an encoding process. Different decisions can be made for different models or for different model implementations, mostly regarding the categorical variables.

3. Evaluation Criteria

When performing internal testing on each model - through the 20/30% validation data, we checked the primary scoring metrics: accuracy, confusion matrix, precision, recall, F1 score, and mean squared error. Accuracy was a good first look at how the model was performing, as it is a simple evaluation of correct predictions out of the total. However, this was not comprehensive enough, especially as we have an imbalance dataset and the model could have a relatively high accuracy with simply projecting the majority class. The next logical step was to produce a classification report and confusion matrix. With these, we could see precision and recall for each of the two predicted values as well as their F1 scores. When testing out values for various parameters in each model, we used the F1 score along with the mean squared error to choose between options resulting in same or similar F1 scores.

Ultimately, we decided to use the F1 score as the main element to decide on our final model for two reasons: 1) Kaggle automatically provided a F1 score with 30% of the test data which gave a better comparison with our validation data F1 score. 2) The F1 score would provide a balance between the previous metrics of precision and recall so its evaluation is more comprehensive.

4. Predictive Models

4.1. Linear & Logistic Regression

We can obtain the trivially attainable performance with only simple models such as linear and logistic regressions. We began with this simpler model to understand the baseline performance level in a quick

and easy manner. This also allowed us to decide how to proceed. Considering that the linear regression produced very high p-values for date, FAQ pages, bounce rate, exit value, browser, and type of visitor, we considered eliminating these features from the analysis or taking a deeper look into how the features may be improved upon. As discussed in feature selection, we ended up by using these features in some models and removing them in others. This was again verified with the statistics in Appendix 1.

4.2.Naïve Bayes

The Gaussian Naïve Bayes model was one of the first we implemented due to its simplicity. It works on Bayes theorem of probability to predict the class of unknown data sets. The drawback being that we're assuming the variables are independent from another. The first thing we did was get the most relevant features. We created a pearson correlation heatmap first. Then took the absolute value of the features that had a greater correlation than .15 to the target feature, Buy. This gave us five different features to work with. One being the 'Season' variable created from 'Date'. Standard scaling proved the most effective scaling method after splitting the data. We figured and were proven correct that the F1 score was well behind other classification models since it assumes that features are unrelated. In reality, it is almost impossible that we get a set of predictors which are completely independent from one another. Here in the techscape ecommerce data set that was proven correct.

4.3. Instance Based KNN Classifier

KNN is a distance based method, so when we take into account that we have multiple categorical attributes whose values don't make sense to be treated in a metric manner (as one is closer to another), we didn't expect KNN to perform really well.

Performing a quick analysis, based on the average training data score, we determined the optimal number of neighbors as 8. The F1 score on the test data set was really low, at 0.089 which led us to not have a more rigorous analysis of it.

4.4.Decision Tree

After the general data pre-processing steps described in Chapter 2 the *DecisionTreeClassifier* was applied with default parameters. This first approach led to a train F1 score of 1 and 0.53 for the validation dataset. Here the Decision Tree Model is clearly overfitting, as we did not specify any parameters the model created a full-grown Decision Tree. This implies that the model will stop running when there are only pure leaves left. However, the corresponding splitting criteria are customized to the training data set. When we applied the predictive model on the validation dataset, the performance decreased.

This is also visible looking at the confusion matrices of the training and validation dataset. The confusion matrix for the validation set shows a considerable amount of false negatives. This can also be concluded when looking at the prediction scores. To prevent the model from overfitting, we altered parameters of the model and created a more robust Decision Tree that is less overfitting.

After completing a baseline decision tree with default parameters, we then tested out other parameters with a more complex pruning method. We decided to go deeper into the decision tree method because of the nature of the business case as we stated in our hypothesis.

First, we tested out whether a gini or entropy criterion would yield better validation results. We also looked at the random splitter but found it to lower the score. Plotting the variables' importance in relation to gini and entropy displayed similar values for each with some variables having more significance with gini and others more significant with entropy. However, the overall test (using F1 score as the primary metric) proved entropy to be a stronger model. This is shown in Appendix C, figures 1 and 2.

Next, we checked various depths of the decision tree with better results with a smaller depth tree. (results in figure C3). A full tree would overfit to the data and not produce a good model. Now to control for overfitting, we needed to check the optimal number of splits for internal nodes as well as leaf nodes. The results of these tests are in figures C4 and C5. According to these tests, one of the best alterations to the model was to set the criterion to entropy and the minimum sample split to 200. The final decision tree after testing parameters is presented in C6.

4.5.Neural Networks

Dissimilar to other classification models, like Instance Based KNN classifier or Naive Bayes Classifier, Multi-Layer Perceptron Classifier (MLPClassifier) depends on a fundamental Neural Network to perform the classification problem. The MLPclassifier method mimics the operations of brains and with its algorithms it is able to recognize profound relationships. With a simple model (no feature selection or customized attributes) it already created relatively good F1 scores between 0.30 - 0.60. After optimizing the attributes for the MLPClassifier it was able to produce F1 scores between 0.59 - 0.67. Improving the score with a neural network model would be time costly and wouldn't substantially improve the model. When performing the model the input is going into a black box, where it performs algorithms and gives outputs based upon these computations. This creates difficulties while optimizing the model. Moreover, it is necessary to give some advice to Techscape. Due to the characteristics of this model it is not possible to give an explanation to outcomes. The combination of a time-consuming optimization process and difficult to explain outcomes, created the decision to focus on other models.

4.6.Ensemble Models: AdaBoost & Gradient Boosting Classifiers

The Adaboost and Gradient Boosting Classifiers used weak learners and built upon those for a more comprehensive algorithm. The primary method we worked on to improve these models was to run functions which passed many different values for each parameter. The resulting plots are shown in Appendix D. We found that even if the score improvement appeared minimal (F1 score of 0.647 to 0.65) during these tests, applying the new parameters in fact made a considerable increase in the final score

(0.67 to 0.7) when combined. The decision to use the default decision tree in adaboost rather than random forest was partially due to the fact that in the decision tree version, each classifier has more or less weight to it depending on its performance - as opposed to the random forest with randomly assigned weights. We also considered the mean squared error on each of these tests. The models were able to learn from their respective mistakes in each previous stump (Adaboost) or tree (Gradient Boost) and performed comparably when run with optimal parameters, both reaching F1 scores of 0.7.

4.7. Bagging Classifier

The Bagging Classifier is another ensemble technique that uses random bootstrap samples from the training data set. Round after round of random bagging samples are processed into classifiers. Bootstrapping being set to true in the model also means that each round, duplicate data is included in the subset. Even though Bagging has a default base estimator of decision tree classifier, we created a separate tree classifier that had the same parameters as our most successful decision tree classifier and assigned it to the base estimator of our Bagging. This gave us a little more control over the model. In addition, we found having a larger number of trees didn't make a difference in the F1 score, when we had our most effective max_leaf_node parameter set to 2. We tried setting bootstrapping to both true and false, but we were slightly more successful without bootstrapping samples.

4.8. Random Forest

For the random forest classifier, dropping the categories that were shown to be redundant on the data exploration proved to be a good efficiency measure, as the F1 score remained very similar. We also experimented in dropping the OS variable as it didn't appear initially to be very impactful on the target variable. This validated our finding in exploration, as the F1 score decreased around 0.1 in the Kaggle performance. While tuning variables in the training set, the number of estimators was determined to be around 200 - although with Kaggle test data, it performed better with 500, according to figure B1. A decision regarding bootstrap had to be made based on Kaggle performance as the training model provided no insight to it (see figure B2). Not using bootstrapping performed 0.02 points better on the F1 score than using bootstrapping. This might be a suggestion that our training data didn't fully represent the space of possible data points. When bootstrapping was considered, the optimal parameters were determined as maximum samples = 80%, maximum tree depth = 12 and the out of bag error rate was minimal at 194 estimators for 'log2' maximum features (see figures B3 through B6). Overall, these models all proved to be producing consistent results in a range of 0.68 - 0.71 F1 score.

4.9. Stacking Classifier

Stacking is another ensemble method we had some success with since it combines the predictions from our best models that we input as base models. From there the predictions are compiled in the meta learning model that best combines the predictions from the base models. It was one of the last models we used as we needed good models otherwise it was so weak it didn't make a difference in our F1 score. The blended model F1 score however was the same as some of our best models. We found it more of a complicated classifier to work with since it's blending multiple models rather than choosing the best one, like a voting classifier. We evaluated combinations of our best models but unfortunately they didn't have a huge impact like we were expecting.

5. Model Comparison & Conclusion

Based on the metrics explained earlier, we narrowed our models down to a few top performers. These models included 1) Decision Tree, 2) Adaboost, 3) Gradient Boost Classifier, and 4) Random Forest without Bootstrap with all four of these models achieving F1 scores between 0.7 - 0.72 considering 30% of the test data.

We faced a trade-off between using a decision tree or a boosting method. We had to compromise either a deeper and more detailed decision tree or an iterating, weighted classifier. One issue we considered when comparing these top models was overfitting. A decision tree was much more likely to overfit to the train data than a random forest. However, this would be marked by a significant difference in the validation and test F1 scores. We noticed similar jumps from validation to test on each of these models which suggested that they were fitting similarly.

With very similar results from the top performing models, we then decided to plot a ROC curve considering only the top four models to better determine each of their prediction abilities - see Appendix G. The metrics obtained through the ROC still showed very similar results among the models, with Gradient Boosting and Random Forest in the lead at 0.928 and 0.927 respectively. We finally decided on the decision tree, as it achieved the highest F1 score in Kaggle with 0.724 with our reasons regarding the importance of F1 score in the section of Evaluation Criteria.

5.1.Conclusion

Our final model for Techscape is the decision tree classifier but we would also present to them models that use the decision tree as it is made to solve this type of binary classification problem. However, we would advise the entrepreneurs to continue to accumulate data as well as possibly expand to other features such as a *Buy* variable for each specific product. This could be fed into a more complex model that helps Techscape focus its business in certain areas of digital detox. In addition, we advise them to return and repeat the analysis as the situation evolves, since the effect of Covid-19 is still present and there may be trends as they accumulate more data.

6. Appendix

Appendix A: Data Exploration & Feature Selection

Figure 1: Count of Access by Month

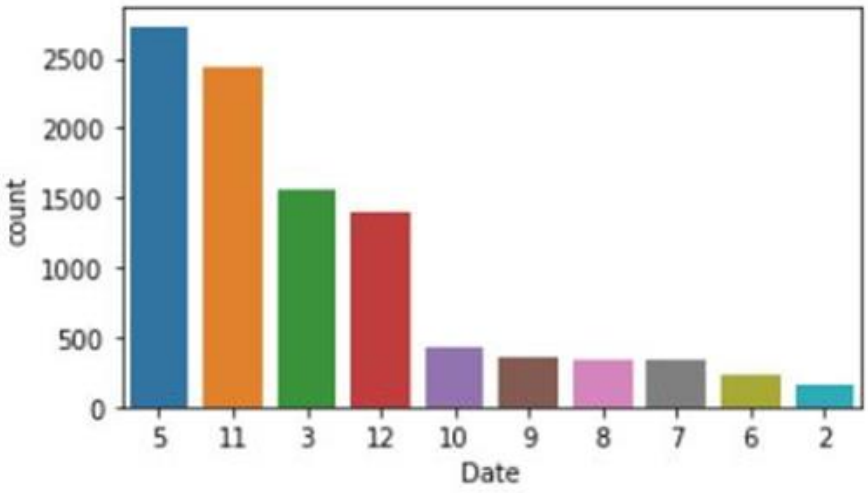


Figure 2: Count of Access by OS Type

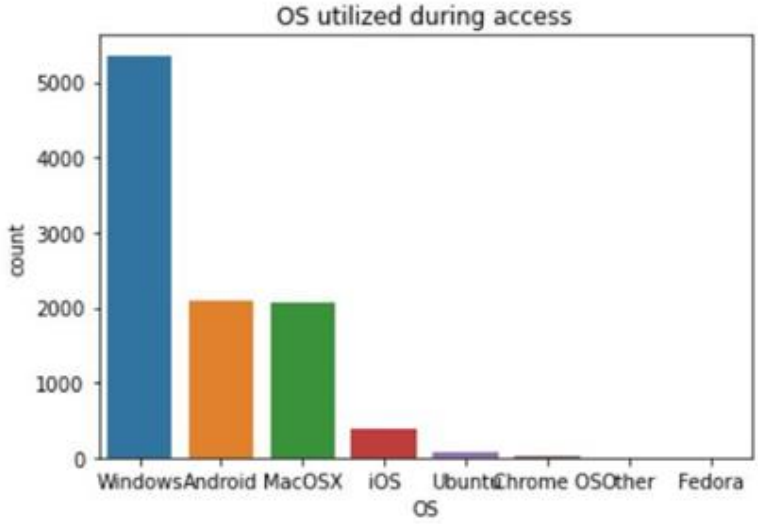


Figure 3: Count of Access by Browser

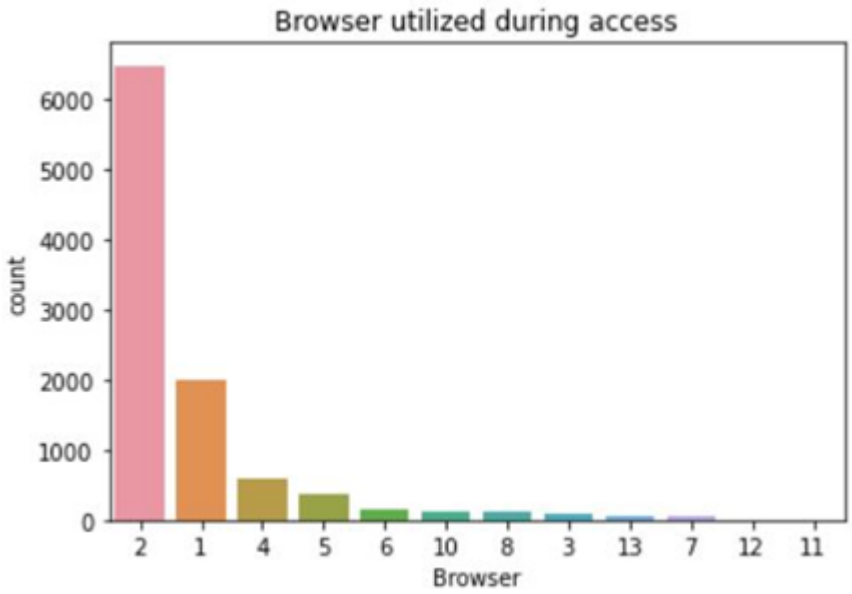


Figure 4: Browser Distribution by OS

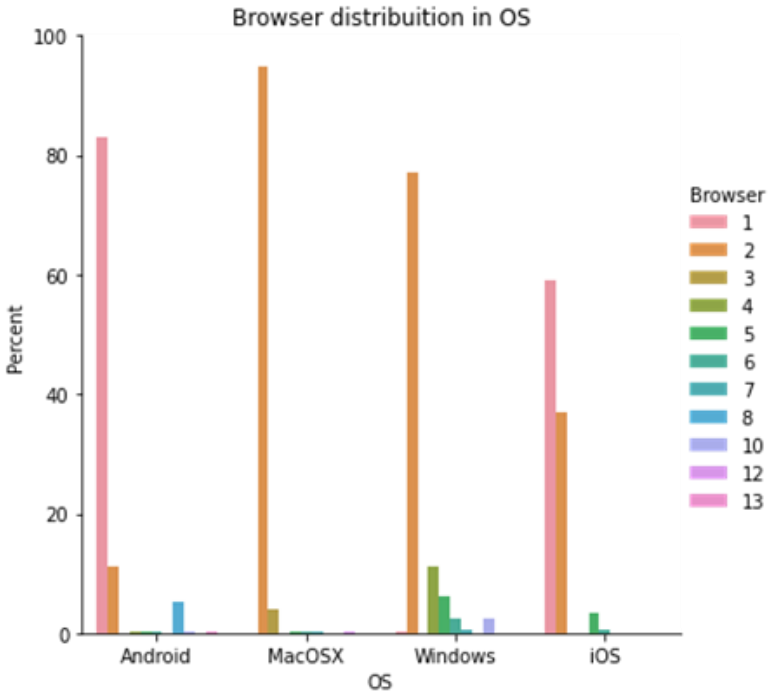


Figure 5: Buy Distribution by Browser

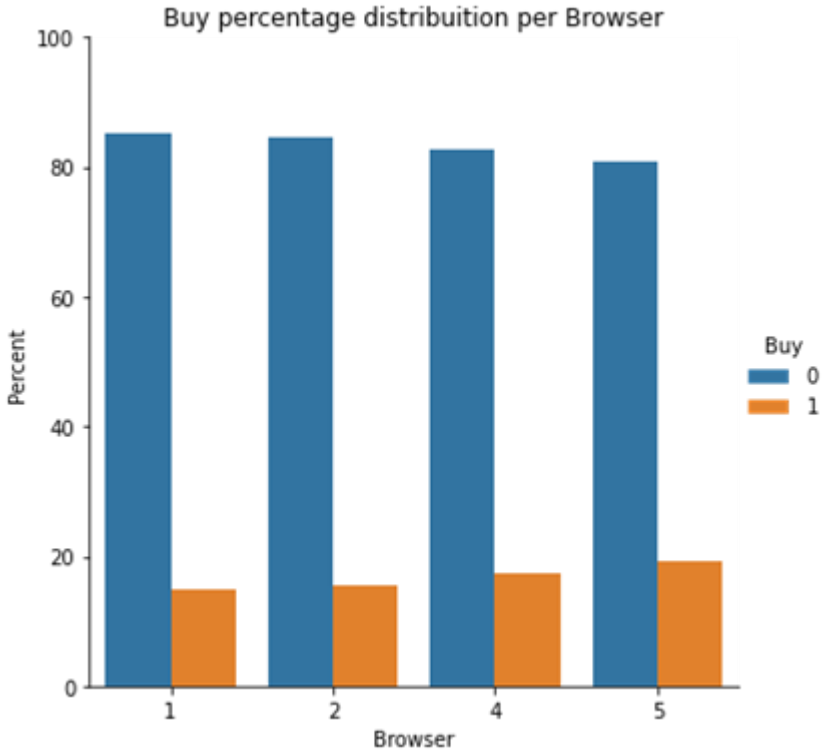


Figure 6: Buy Distribution by OS

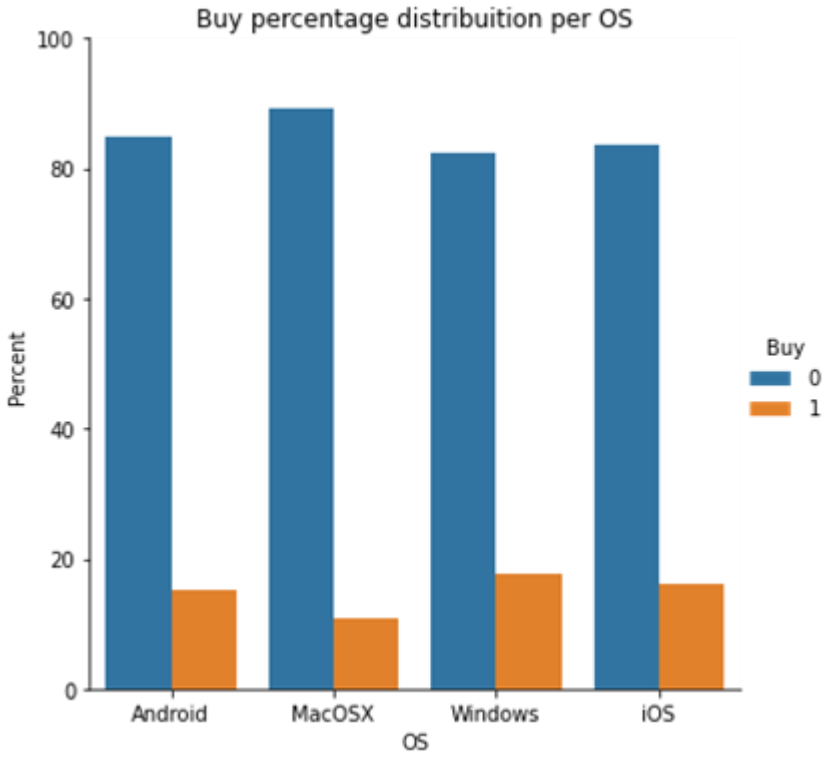


Figure 7: Buy by Type of Visitor

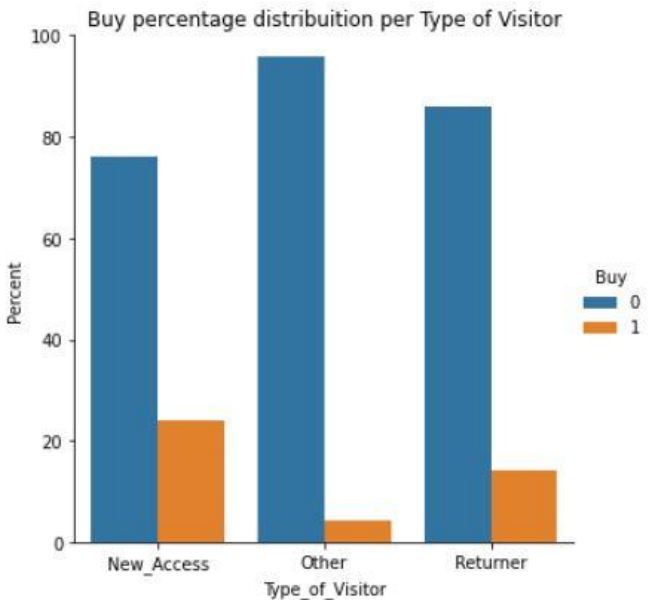


Figure 8: Buy Percentage by Type of Traffic

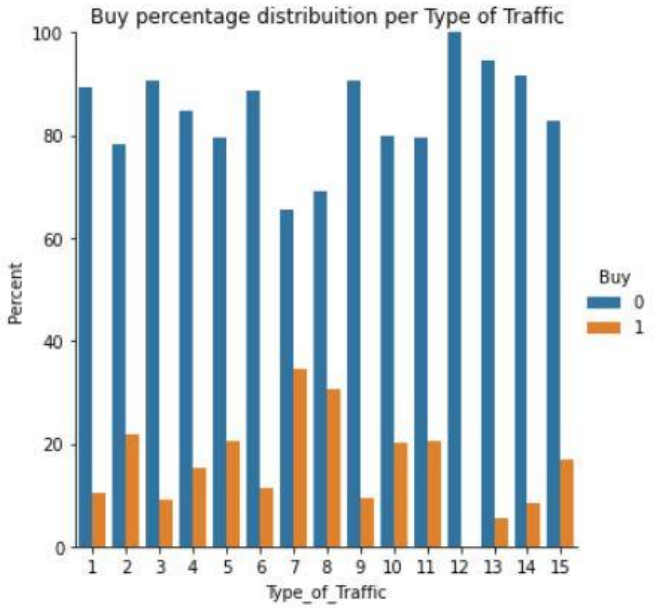


Figure 9: Correlation Matrix for Numerical Variables

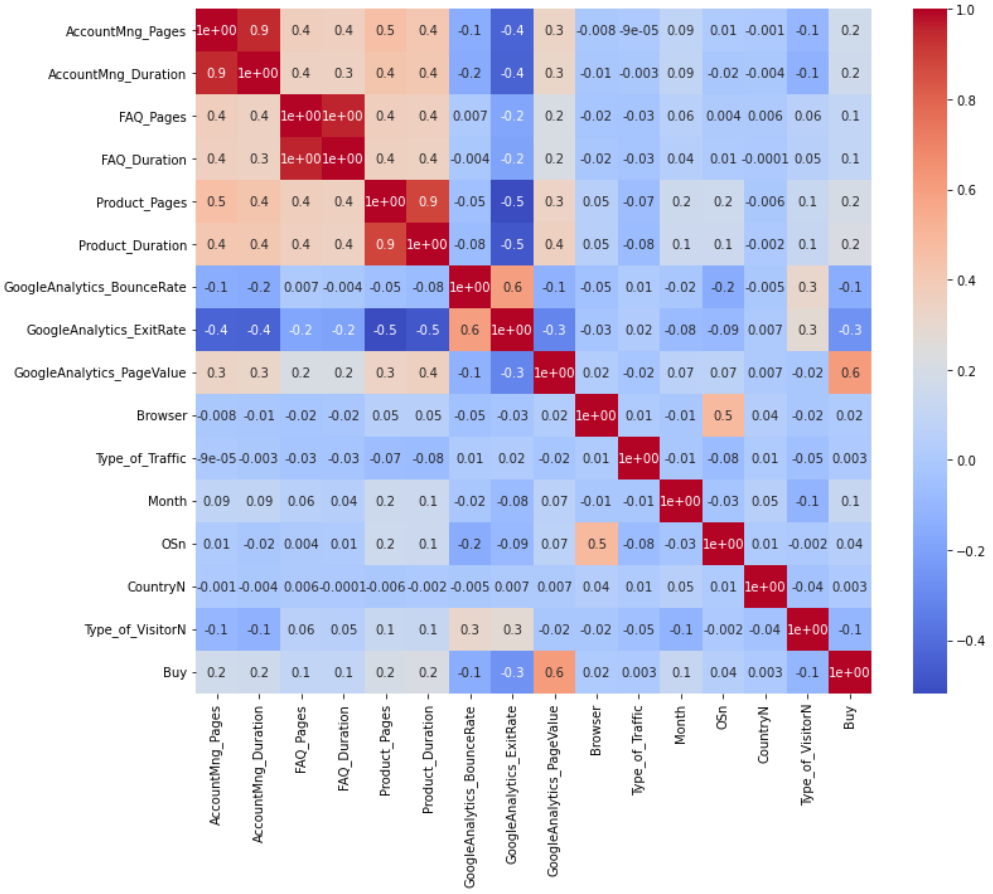
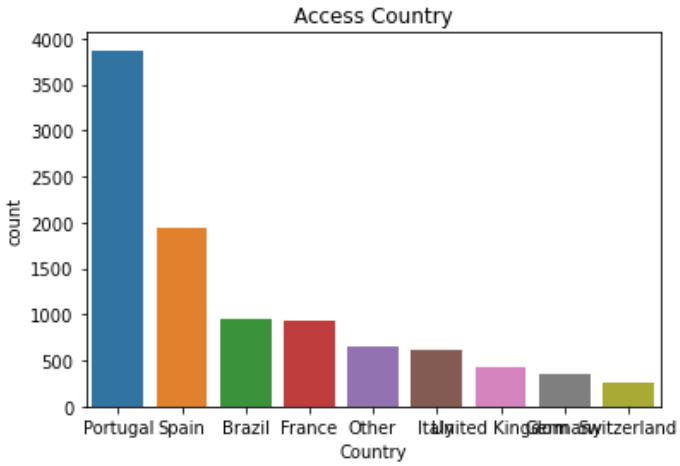


Figure 10: Access count by Country



Appendix B: Linear Regression

Figure 1: Descriptive statistics

```
Residuals:
    Min      1Q  Median      3Q     Max
0.0074 0.0615  0.1061 0.1643 3.4218


Coefficients:
                              Estimate          Std. Error          t value   p value
_intercept                   -5.227565  0.639344+0.000000j  -8.1765+0.0000j  0.000000
Date                          0.000000  0.000000+0.000000j   0.0000-0.0000j  1.000000
AccountMng_Pages              0.003397  0.001418+0.000000j   2.3961-0.0000j  0.016592
AccountMng_Duration           0.000017  0.000021+0.000000j   0.8051-0.0000j  0.420778
FAQ_Pages                     0.000172  0.003736+0.000000j   0.0461-0.0000j  0.963241
FAQ_Duration                  0.000004  0.000028+0.000000j   0.1385-0.0000j  0.889852
Product_Pages                 0.000325  0.000165+0.000000j   1.9648-0.0000j  0.049469
Product_Duration              0.000012  0.000002+0.000000j   5.9768-0.0000j  0.000000
GoogleAnalytics_BounceRate -0.000013  0.152242+0.000000j  -0.0001+0.0000j  0.999932
GoogleAnalytics_ExitRate   -0.000017  0.163129+0.000000j  -0.0001+0.0000j  0.999918
GoogleAnalytics_PageValue   0.009115  0.000187-0.000000j  48.6792+0.0000j  0.000000
Browser                       0.000225  0.002135+0.000000j   0.1052-0.0000j  0.916225
Type_of_Traffic              -0.001355  0.000956-0.000000j  -1.4180-0.0000j  0.156219
OSn                           0.001291  0.001466-0.000000j   0.8807+0.0000j  0.378514
CountryN                     -0.002011  0.001630-0.000000j  -1.2339-0.0000j  0.217273
Type_of_VisitorN             -0.000071  0.005154-0.000000j  -0.0138-0.0000j  0.988960
---
R-squared:  0.26003,    Adjusted R-squared:  0.25864
F-statistic: 187.02 on 15 features
```

Appendix C: Decision Tree

Figure 1: Checking the importance of each feature with gini and entropy criterions

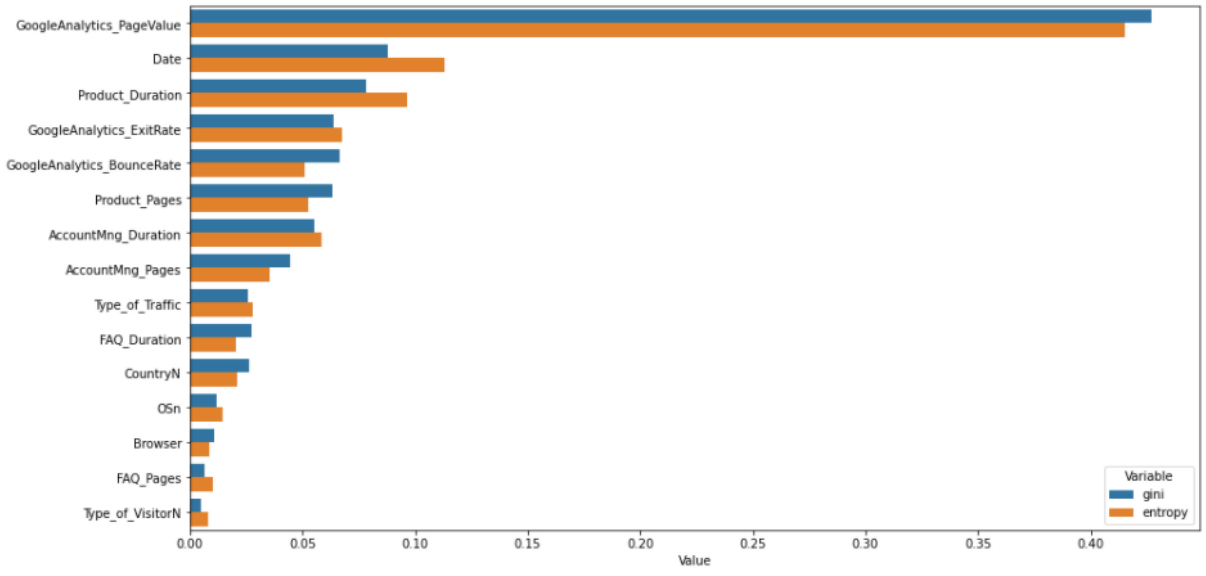


Figure 2: Checking the importance of each feature with gini and entropy criterions

	Time	Train	Test
Gini	0.068+/-0.0	1.0+/-0.0	0.824+/-0.01
Entropy	0.113+/-0.02	1.0+/-0.0	0.831+/-0.01

Figure 3: Checking depth of tree

	Time	Train	Test
full	0.086+/-0.01	1.0+/-0.0	0.825+/-0.01
depth2	0.016+/-0.0	0.854+/-0.0	0.854+/-0.0

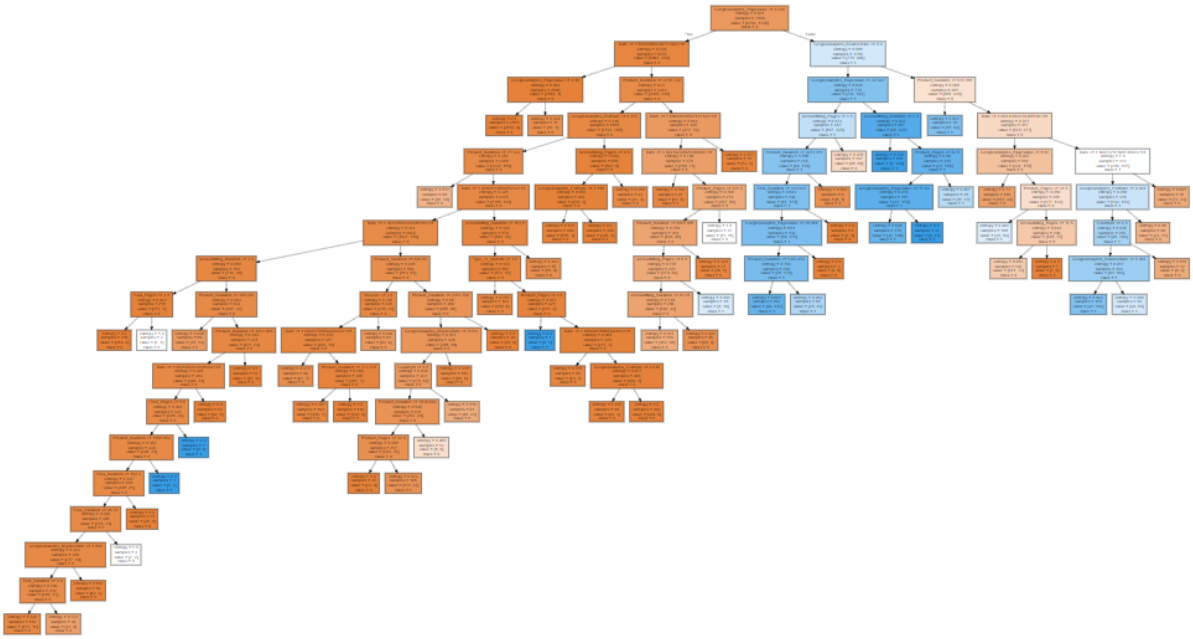
Figure 4: Checking minimum samples split of tree

	Time	Train	Test
dt_min200	0.05+/-0.0	0.882+/-0.0	0.869+/-0.01
dt_min700	0.041+/-0.0	0.867+/-0.0	0.857+/-0.01

Figure 5: Checking minimum samples leaf of tree

	Time	Train	Test
Original	0.073+/-0.01	1.0+/-0.0	0.824+/-0.01
dt_min_sam100	0.041+/-0.0	0.871+/-0.0	0.861+/-0.01
dt_min_sam500	0.025+/-0.0	0.859+/-0.0	0.855+/-0.01

Figure 6: Final model



Appendix D: Ensemble Methods

Figure 1: Checking scores for maximum depth values with a decision tree

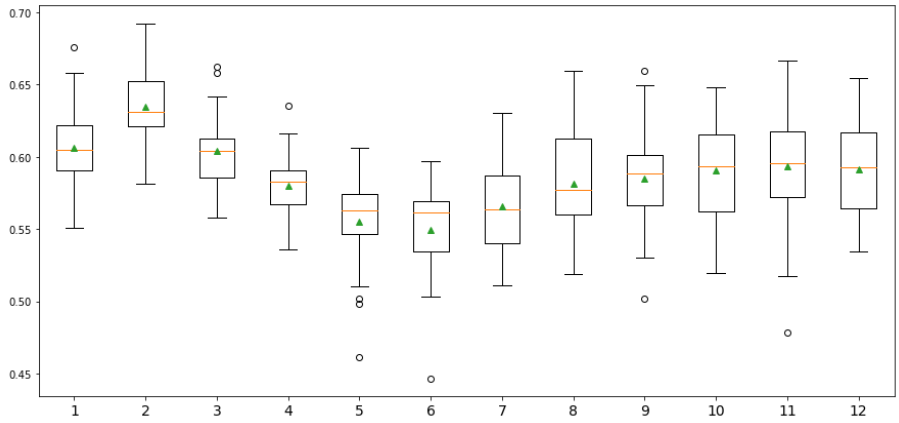


Figure 2: Checking scores for maximum depth values in a random forest

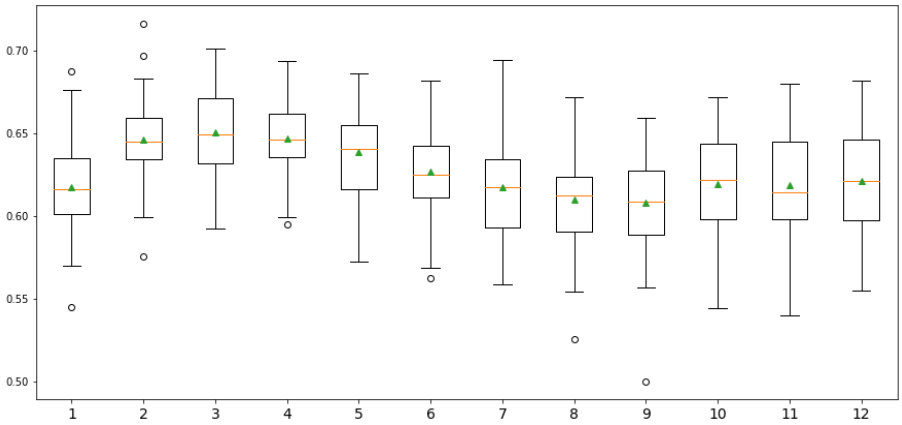


Figure 3: Checking scores for learning rate values with a decision tree

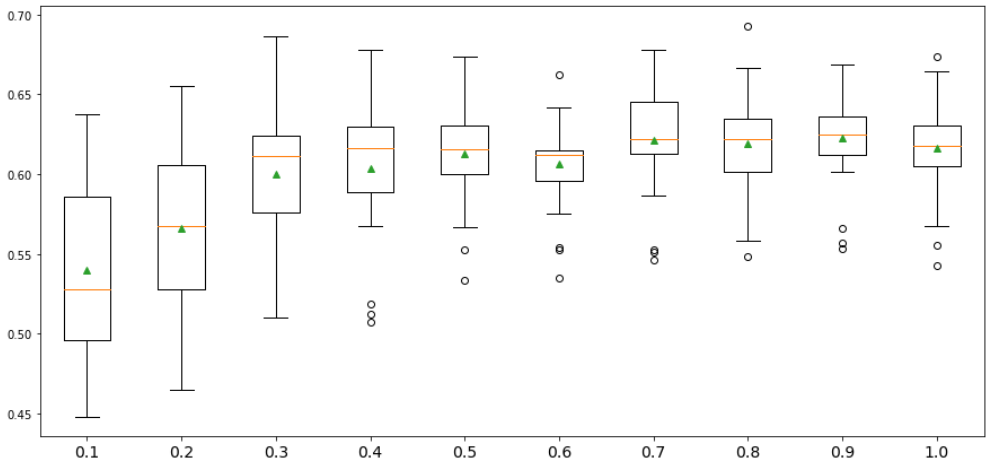
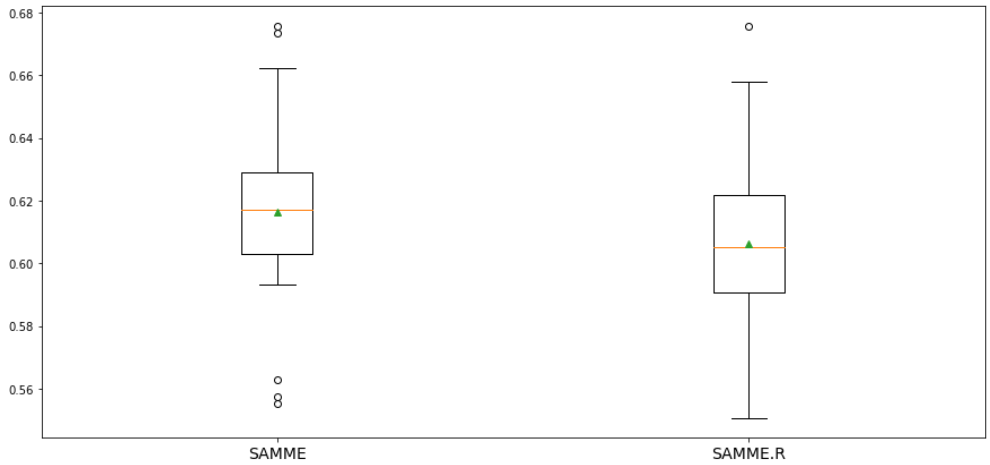


Figure 4: Checking scores for algorithm SAMME vs SAMME.R



Appendix E: Random Forest

Figure 1: Checking scores for number of estimators parameter

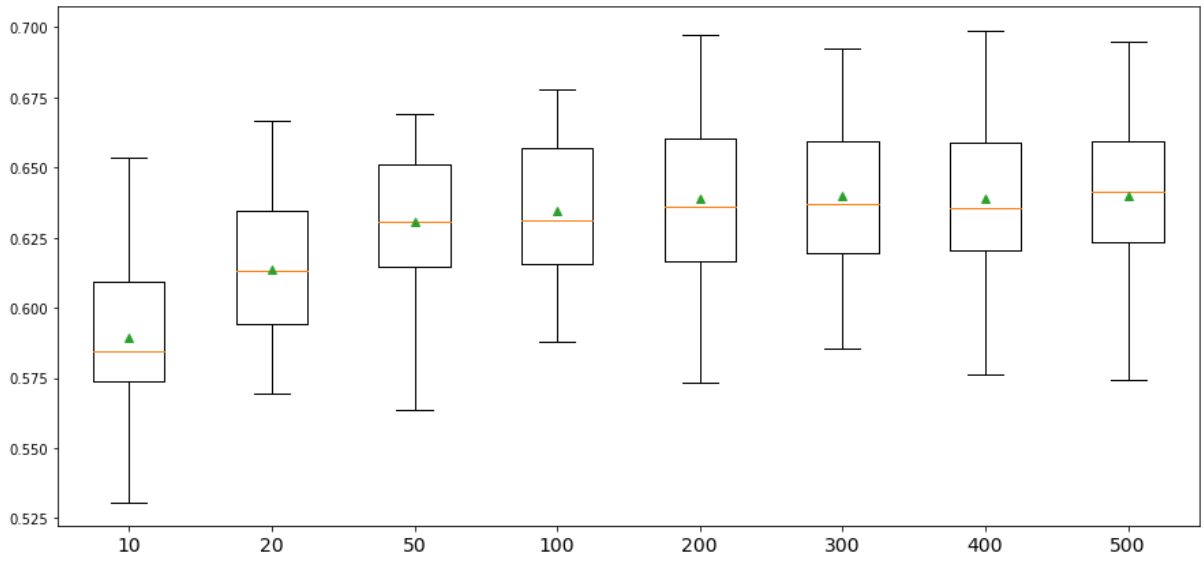


Figure 2: Checking scores for bootstrapping parameter

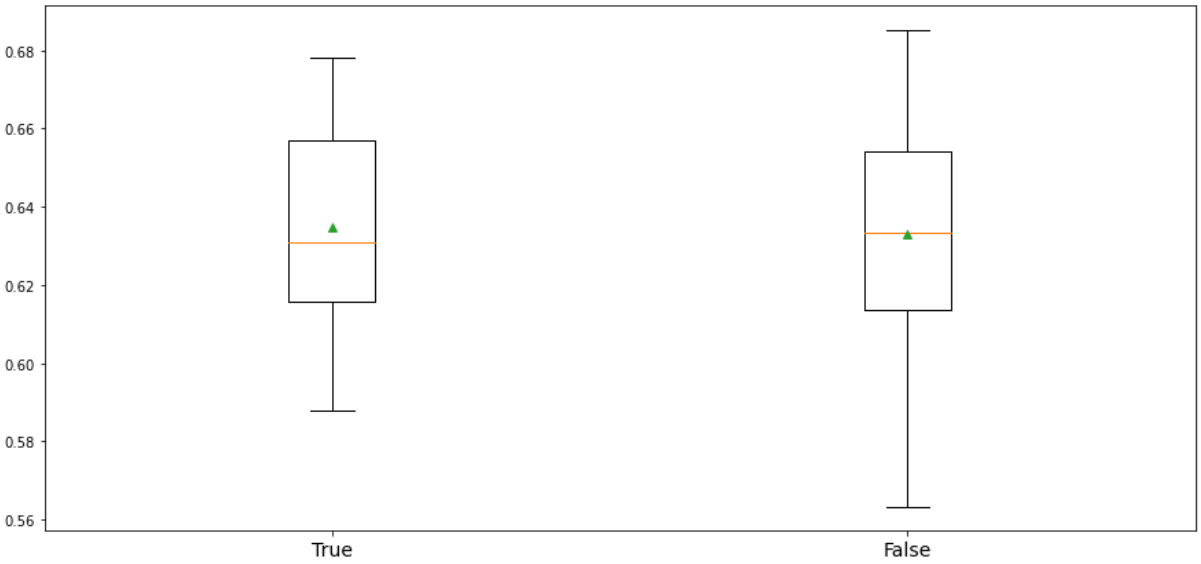


Figure 3: Checking scores for bag sample in bootstrapping parameter

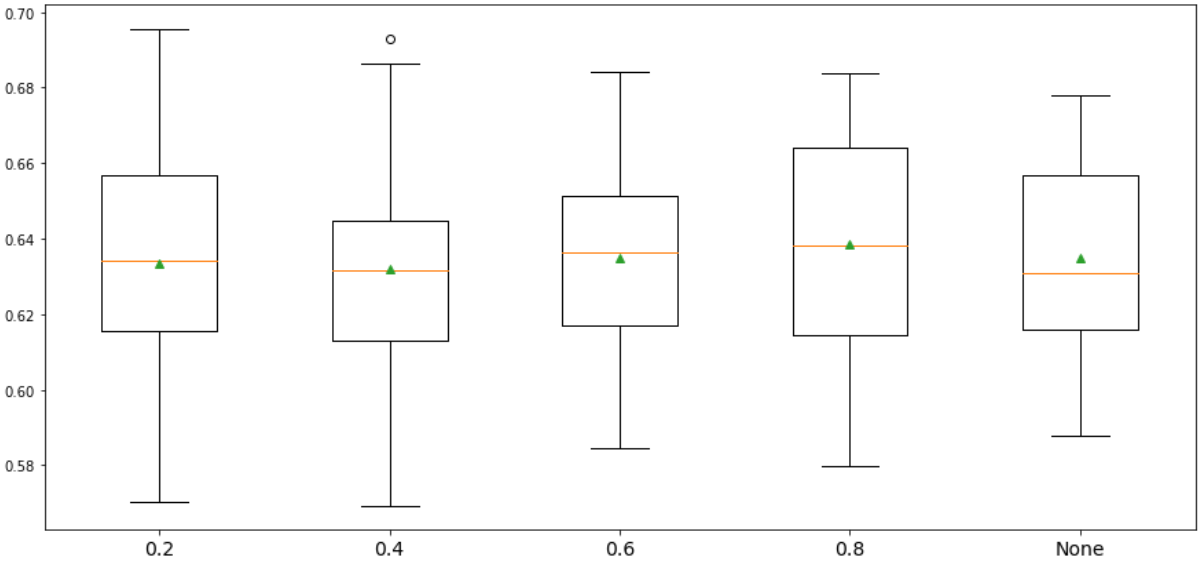


Figure 4: Checking scores for tree depth parameter

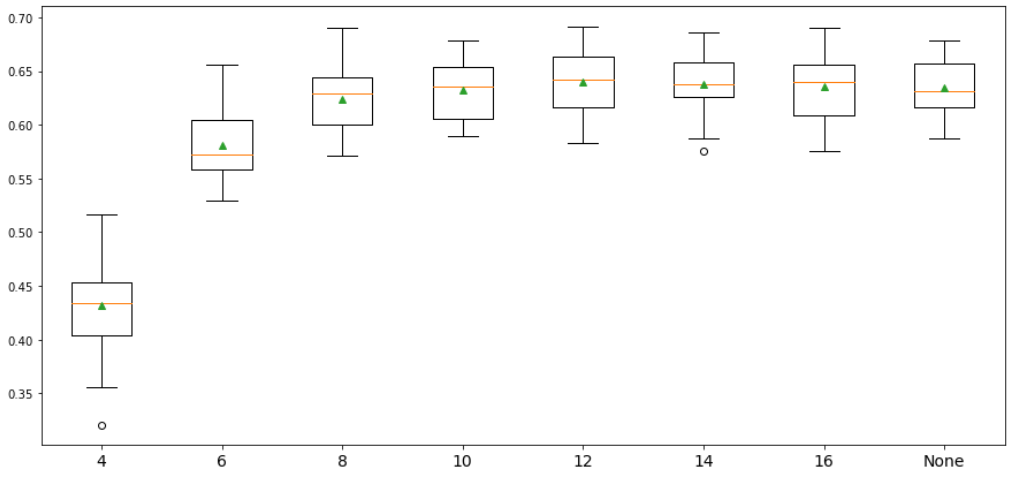
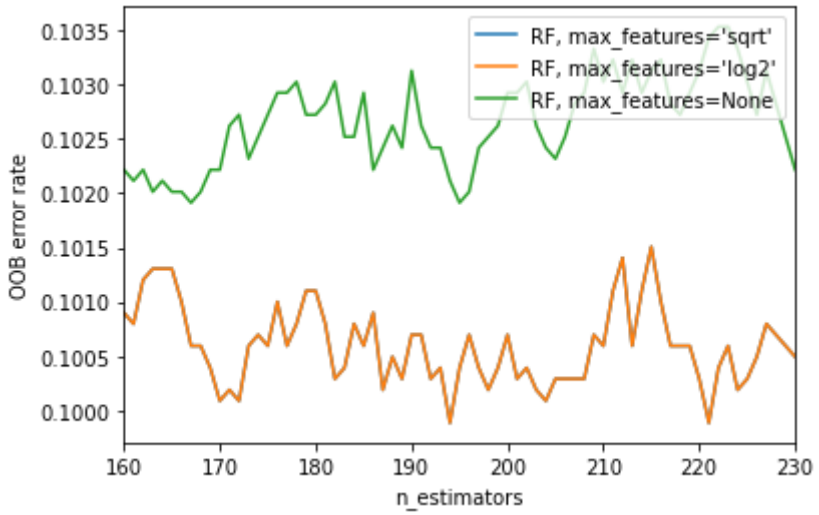
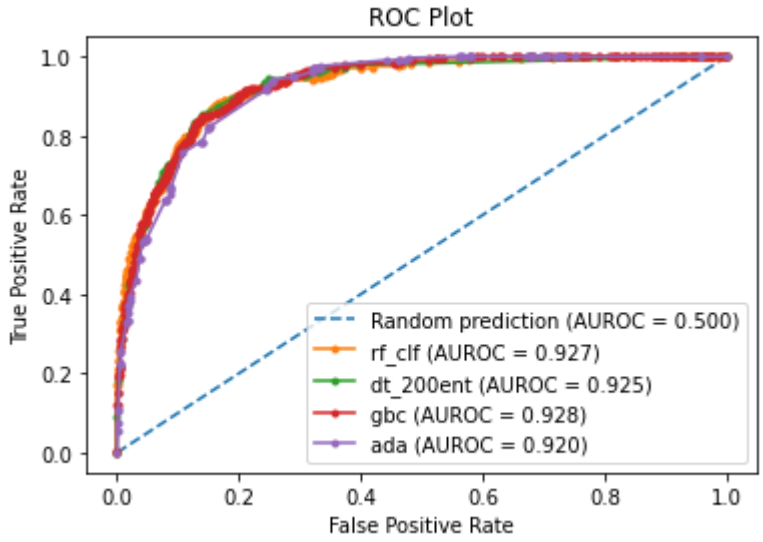


Figure 5: Checking error rate with maximum number of features parameter



Appendix F: Comparison of Models

Figure 1: Receiver Operating Characteristic (ROC) Curve



6.1.Appendix G: Additional Models

Voting Classifier

Once we had several accurate models with a high prediction score, we decided to use the Voting Classifier from the sklearn ensemble library. We discovered it helped when we gave the Voting Classifier a variety of different models. Our thought process was it would be more effective to input the findings of our most effective models into one, and have it predict the output based on the combined majority of voting for each output class. We learned that hard voting predicts the class with the largest sum of votes, as opposed to soft voting, which predicts the class with the largest summed probability. After that, we had more success with hard voting. An obvious limitation we found though, is that the voting classifier uses all the models equally. Since some models like our Gradient Boosting Classifier were more effective, we would have to have weighted voting to combat the problem. We chose not to pursue that route though as we had more success with other classifiers.

Histogram Gradient Boosting Classifier

Histogram based gradient boosting technique works similar to the regular Gradient boosting method except it uses binning to put the values into buckets. So it works much faster as it reduces the number of unique values. The F1 score was still around the same score as the original Gradient Boosting Classifier which makes sense. Since it was faster than regular Gradient to have a high number of iterations and a low learning rate, we experimented with it to give it robustness. We also experimented with the model as an input model for the Voting and Stacking Classifiers since it's significantly faster.

7. Bibliography

Saini, Anshul. “Gradient Boosting Algorithm: A Complete Guide for Beginners.” *Analytics Vidhya*, 20 Sept. 2021, https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/.

Saxena, Shipra. “AdaBoost Algorithm: AdaBoost Algorithm Python Implementation.” *Analytics Vidhya*, 26 Mar. 2021, https://www.analyticsvidhya.com/blog/2021/03/introduction-to-adaboost-algorithm-with-python-implementation/.

“Sklearn.ensemble.gradientboostingregressor.” *Scikit*, 2021, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html.

“1.5 Stochastic Gradient Descent” *Scikit,* 2021, https://scikit-learn.org/stable/modules/sgd.html.

“Sklearn.ensemble.StackingClassifier.” *Scikit*, 2021, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html

“Sklearn.ensemble.VotingClassifier.” *Scikit*, 2021, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

“Sklearn.ensemble.HistGradientBoostingClassifier.” *Scikit*, 2021, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html#sklearn.ensemble.HistGradientBoostingClassifier.

“Sklearn.ensemble.GradientBoostingClassifier.” *Scikit*, 2021, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

“Sklearn.ensemble.RandomForestClassifier.” *Scikit*, 2021, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

“Sklearn.ensemble.BaggingClassifier.” *Scikit*, 2021, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html.

Brownlee, Jason. “Stacking Ensemble Machine Learning With Python.” April 27, 2021, https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/.

Mirjalili, Vahid. “Python Machine Learning.” 2019, Packt

“Sklearn.metrics.roc_curve” *Scikit*, 2021, https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

Guyon, I., Elisseeff, A., & Kaelbling, L. P. 2003, ”An introduction to variable and feature selection.” Journal of Machine Learning Research, 3(7-8), 1157–1182

Géron A., 2019, “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems”, O’reilly Media