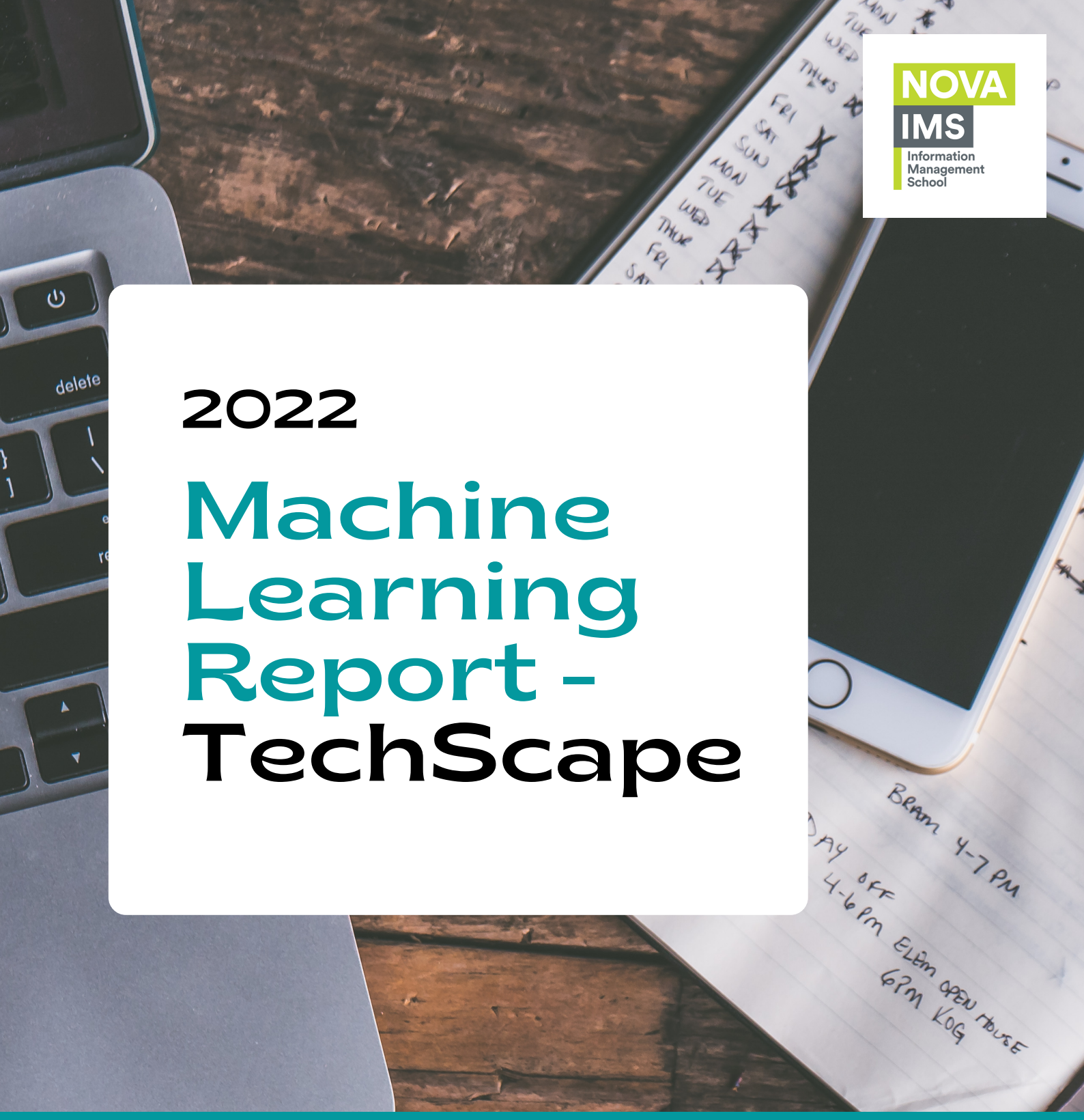

NOVA IMS Information Management School

2022

Machine Learning Report - TechScape

GROUP 11

Sarra Jebali | 20210765
Yuri Ferreira | 20180427
Pauline Richard | 20211018
Opeyemi Mary Akande | 20211320
Tiago Ferreira | 20211317

Summary

Abstract

This project was made to predict the customer purchasing behavior of the online platform, TechScape, through a machine learning approach. We wanted to determine which customers were more likely to buy the products, with two possible outcomes: Customers are likely to be buyers (1) or not (0).

For this purpose, we used the continuous customer information monitored by the startup for 10 months - from February until December 2020, excluding April. The data are accessible in a Train dataset and a Test. Our end goal was to obtain the predictions on the test dataset with the highest f1_score possible. Therefore, first, we extracted the observations. Then, we checked their quality, to make sure they are efficient for a Machine Learning application. Finally, after pre-processing the data and selecting the relevant ones, we worked on several machine learning models for prediction. And we came to the conclusion that the Random Forest was the best fit for the dataset, and an f1_score of 72.18% was obtained.

Introduction

Beginning of the year 2020 was launched the new Portuguese startup: TechScape. In a more and more technologized world, the purpose of the company was to help people to reconnect to themselves by promoting digital detox. Unfortunately, the worldwide pandemic of COVID-19 arose at the same time, resulting in financial difficulties and a break of one month in activities of the company, in April 2020.

They are now back on track since May 2020 with a new promising niche market, looking for a digital detox. By analyzing their database, we were aiming to identify and predict this niche, and therefore help TechScape increase their sales. We will first try to understand the variables, their relationships, and their different patterns in order to, in a second part, work on a solid base and build a predictive model with the best accuracy possible.

The Database consists of a training dataset of 9 999 observations, with a target variable “Buy”. when the target is equal to 1, the client is likely to buy a product, and when is 0, the client is not likely to buy anything. The observations are based on the online actions of the customers on the website such as time spent on different pages, pages seen, exit rates, etc… And also some specific information such as the type of visitor, which browser they are using, in which country they live in, etc. We have a total of 17 variables. All our prediction attempts will be performed on the Test dataset, made of 2 300 observations but with no target variable, and using the pattern observed in the Train.

1. Let's Explore and analyze

Before we can test a model, we need first to understand our data and try to get as many insights as possible. In this first part, we are going to check the quality of our data, the relationships and patterns between the variables, but also spot anomalies and test hypotheses using visualization tools.

1.1. Data Description

Our Train dataset is composed of 9 999 rows and 17 columns. The types of variables are either integer, float, or objects.

Our target variable is "Buy" and has two possible outcomes, as mentioned before:

- 8447 observations represent users that did not buy something on the website
- 1552 observations belong to users that finalized their actions with a transaction on the website

We can see that the two options are not equally represented, the outcome "0" represents 84.47% of the dataset, while "1" represents the 15.52% remaining. We are therefore clearly dealing with an imbalanced dataset that we will have to deal with in the next part (Data Preparation).

Moreover, many variables share the same type, scale, and characteristics, and we can imagine we will have a lot of strongly correlated variables. We can take as an example the variables '**GoogleAnalytics_BounceRate'** and '**GoogleAnalytics_ExitRate'** where the difference between the two is subtle, and they are giving quite similar information.

1.2. Assessing Data Quality

The quality of our Train dataset is going to have an important impact on the accuracy and efficiency of our model, so let's have a deeper look into this.

1.2.1. Checking for missing values and duplicates:

First, we checked for missing values and strange values such as (!, $, %, ?, *, +, _, @, €), which can be the result of mistakes or failure to record the data. After using train.isna, and checking for the strange characters, we can conclude that there are no missing values neither in the training dataset nor on the test dataset. We also did the same to check duplicates and came to the same conclusion.

1.2.2. Checking for variables' correlation:

Secondly, we had a look at the correlation between our variables using a heatmap. We observed that some variables are highly correlated For instance:

- AccountMng_Pages and AccountMng_Duration
- FAQ_Pages and FAQ_Duration
- Product_Pages and Product_Duration

- GoogleAnalytics_BounceRate and GoogleAnalytics_ExitRate

1.2.3. Checking for outliers:

The next step was to check if the dataset contains any outliers. Our first approach was to study the dataset descriptive table. By analyzing the values of the min, mean, and max of each variable, we noticed that while there is almost no difference between the mean the min values of the different features, suggesting that there are no low outliers, the mean and max show huge differences in values for all the features which indicates that we have a problem of extreme values for all the variables. Our second approach was to build boxplots for all the variables. Commonly, we consider outliers a data point that is numerically distant from the rest of the data, here defined when it is more than 1.5 IQR above the third quartile or below the first quartile, with IQR= Q3-Q1. Looking at the outcome of our box plots, we could see all of our variables seem to contain outliers with extreme values above the upper quartile. Finally, we were able to confirm the existence of extreme values using the IQR method and the z-score.

1.2.4. Coherence Check:

In the final step of assessing the data quality, we decided to run a coherence check. We wanted to make sure that the data is logically consistent and can be reliably combined for analysis. During our analysis, we noticed that some values do not make sense. One problem that occurred frequently was that some observations suggested that a client opened a website page (for example, Product_Pages=5) and spent no time on those pages (Product_Duration=0), which is inconsistent and impossible. We figured that in order to build a reliable model, we needed to find a solution for this issue.

Conclusion: After carefully exploring the dataset, we can conclude that it does not contain any missing values or duplicates. Except for the outliers that will be dealt with later in this project, we can classify this dataset as satisfactory and appropriate to achieve the project's objectives.

1.3 Data Visualization

To explore and understand our dataset better, we also used some Data visualization tools. By exploring the variables through different bar charts and plots, we were able to identify patterns and extract two major pieces of information.

First, looking at the boxplots, besides the outliers' information, we can observe additional details. We can see by comparing the interquartile ranges, that the length of all the box plots is small, showing a small dispersion of the data. We can also see that all the variables are showing a positively skewed distribution, with the median closer to the bottom of the box. It means that the average of all the values (mean) is greater than the middle values of the data (median), and can be linked with the observations we did of the outliers.

In a second part, we analyzed the relevancy of our categorical variables with our target, using the function cross tab of Pandas and visualizing the summary with a bar chart. Therefore, we were able to conclude that the categorical variable 'Country' was not very relevant to our model, since regardless of the value taken, the 'Buy' or 'not Buy' proportion was quite the same for all different values. However, variables such as 'OS', 'Type_of_traffic', or 'Browser' are giving important information. For example, for the latest one, the target outcome is more frequently '1' when the Browser used is the 13, and on the contrary, it is always '0' when the client is using the Browser with value 11.

2. Let's Prepare the database

During this phase, we will be manipulating and transforming data elements into a useful and decent dataset to match the needs of the different models that will be built.
In the steps that follow, our data preparation analysis was performed on the training dataset, and then applied to the test dataset. This was made to ensure that when we use the test dataset to make the predictions, the model will treat it as new information seen for the first time. and the score will be a better representation of the results we want to reach.

2.1 Coherence check

To fix the problem of the inconsistent data values, we thought of replacing those incoherent numbers without messing with the overall mean of each column. To achieve this, we decided to replace the 0s in the duration columns (when they shouldn't be 0) with the mean duration for each column depending on the number of pages seen. We first calculated the mean for each group of pages visited for the 3 columns. We then created a for loop that will change the 0 in the duration depending on the value in the number of pages visited in both the train and test dataset. The result reached was a more balanced coherent dataset that gave a better score in model assessment.

2.2 Outliers

As seen in the first part, we have outliers to deal with. However, it is important to mention that determining whether a value is an outlier that should be removed or not is very subjective. And while there are certainly valid reasons for throwing away outliers if they are the result of a computer glitch or a human error, eliminating every extreme value is not always a good idea. In our case, for instance, some outliers are unusual values and impossible to occur. The column of 'Project_duration' has some values exceeding 60000sec, suggesting that a person spent more than 17 hours a day going through the project page, which is not likely to happen. At the same time, some values can present significant information about the data. The extreme values in the column of 'GoogleAnalytics_PageValue', for example, represent a higher percentage of buyers than non-buyers, despite the fact that our dataset is unbalanced. A better representation of this phenomenon is seen in the table in Figure 2 (Annexes).

We notice that when the value of ‘Buy’ is 1, the mean for each variable is bigger and sometimes even largely bigger, which emphasizes that some large values could help us identify the buyers. Therefore, We decided to keep some extreme values for further investigation on their meaning and deal with the ones that look inaccurate.

This being said, we tried different options to reduce or remove the outliers:

- The IQR method: not possible since some features have an IQR of 0
- The filter method: not very optimal since filtering is done manually
- The Z-score method couldn’t be used since we are dealing with a positively skewed distribution
- Quantile-based Flooring and Capping was a better approach. We calculated the 99th percentile for each column and decided to replace the values that were bigger than it, with that value instead. This method showed a better score.
- Square Root Transformation: we used this method to be able to keep some extreme values and at the same time reduce their effects in the dataset and make all the values relatively closer. This approach also gave a better score.

Finally, we decided to use the combination of both methods: Quantile-based Flooring and Capping & Square Root Transformation.

2.3 Imbalance Dataset

Since we are dealing with an imbalanced dataset resulting in a significantly lower F1-score for the minority label, we decided to try and balance the data by trying two methods:

- SMOTE
- Random oversampling and undersampling

Unfortunately, both of these methods did not increase our final f_score but lowered it even more.

2.4. Feature engineering

Valuable insights can be derived from the ‘Date’ variable. However, it is not possible to implement that variable the way it is. For this reason, we decided to exploit it by dividing it into two columns: ‘Day’ and ‘Month’ which will respectively have the day and month of that date. We did not create a column for the year because our dataset captures observations occurring in 2020 only, and therefore that column will have a variance of 0.

It is worth mentioning that we tried to create new variables that will minimize the number of features and at the same time capture the same amount of information. We sought to create 3 new variables that will calculate the time spent for each of the 3 pages available by dividing the duration by the number of pages seen. Unfortunately, this did not improve our f1_score, and we decided to eliminate this idea.

2.5. One hot encoding:

Since most models can not work with categorical variables, this step becomes necessary. One-hot encoding is the process of changing categorical data into numerical ones. And when dealing with this step, the two, most common options that might be considered are:

- pandas.get_dummies: This function turns categorical values into dummy values, meaning binary values of either 0 or 1. However, this method creates a column for each option in the categorical variable.
- pandas.factorize: This method is useful for obtaining a numeric representation of an array when all that matters is identifying distinct values, giving each option a numeric value. The drawback of this method is that the algorithm might give more importance to options with the highest number.

Since the categorical variables in this dataset are not ordinal, and we do not want our models to prioritize certain observations compared to others, we decided to use the get_dummies method. However, before applying it, and because we do not want to create extra columns for non_important variables, we decided to run a feature selection method on categorical features to decide which ones will be kept and apply the one-hot encoding in those variables only.

2.6. Data Split:

For this step, we will separate the data into features and targets. The training data will be mixed and randomly split into a training set composed of 85% of the data, and a validation set, representing 15% of the data. We chose to work with a validation set beside the test set to be able to tune the parameters of a classifier and evaluate the performance of models for different combinations of hyperparameter values.

Moreover, considering we have an imbalanced dataset, we were careful to implement the stratified sampling option, so we ensure that the training and the validation set have both the same proportion of each class as the original dataset. Once the best parameters are chosen for each model, we will use the test set to compare the different performances.

2.7. Normalizing the Data:

Normalizing the data is an important step in data preparation, especially if we're going to deal with models that rely on the distance between the different observations. For this dataset, we tried 4 different approaches:

- MinMax scaler [0,1]
- MinMax scaler [-1,1]
- Standard scaler
- Robust scaler

The MinMax scalers did not work well with the models, which makes sense since this scaler is very sensitive to outliers. On the other hand, the standard scaler, despite being sensitive to outliers as well,

showed better results when compared to the robust scaler, a model that is not influenced by extreme values.

2.8 Features Selection

Not all the features present in a dataset are useful in building a machine learning model. Sometimes using all the variables can be time-consuming, confusing, and impractical for the model. For instance, it is not useful to build a model using a dataset containing many strongly correlated features. And some other features may be a noise that decreases the performance by confusing the model and giving it irrelevant data that prevents it from learning the actual relationships. For these reasons, we will run different feature selection methods to decide which variables to keep and which ones to remove. The methods that will be applied will help us decrease the number of features while keeping the same amount of information.

2.8.1. Categorical Data:

To determine the important features among categorical variables, we used the Chi-square method. After running the algorithm, we got conflicting results to the hypothesis we built earlier in the data visualization phase. Before, we have concluded that the 'Browser' variable is an important feature and it was visible the buyers' proportions differences depending on the browser used. However, this method showed that the 'Browser' variable has no importance in the modeling process. For this reason, we will run the model with and without it and observe the difference in the results.

2.8.2. Numerical Data:

To determine the useful numeric data to our model, we followed two approaches:

- **Redundancy**: we checked if some features are giving the exact same information to the model, for that we used a heatmap, as shown in Figure 1 (Annexe)

After analyzing it, we determined that there were four pairs of highly correlated variables, so there would need to be eliminated either AccountMng_Pages or AccountMng_Duration, FAQ_Pages, or FAQ_Duration, Product_Pages or Product_Duration and finally, we would have to keep either GoogleAnalytics_BounceRate or GoogleAnalytics_ExitRate.

- **Relevancy**: How much the variables are actually relevant in predicting our target variable.

To determine which features were more important to the model, we used filter, wrapper, and embedded methods. Four different methods were used: First, we used the correlation to check which variables are highly correlated with our target variable. Then we used recursive feature elimination: RFE. Finally, Lasso Regression and decision tree importances were applied. These two methods are not only important in determining which variables are important and which are not, they are also helpful in choosing which one of the correlated variables we should remove. Since these models will

give a score of importance to each variable, out of the 2 correlated features, we will keep the one with the highest score.

The results of all the methods are summarized in Figure 3 (Annexes).

However, since these models gave us conflicting results, we decided to run different combinations of features with different models and compare the scores each time. The final variables chosen were: AccountMng_Duration, FAQ_Duration, Product_Duration, GoogleAnalytics_ExitRate, GoogleAnalytics_PageValue, Month, Day, OS, Type_of-Trafic, and Type_of_Visitor.

3. Let's Predict with Models

Our Train dataset is now prepared with optimal features and adjusted data that are going to feed our model. For this last phase, we are going to see how we applied and improved many modeling algorithms. Since predictive algorithms can work for the same problem, we tried different techniques and selected the best one.

The models used were the following:

- K-nearest neighbor
- Logistic Regression
- Support Vector Machine
- Decision Tree
- Neural Networks
- Quadratic Discriminant Analysis
- Random Forest
- Gradient Boost Classifier
- Adaboost Classifier

3.1 Model Parameters:

Our first approach was to try the models with their default parameters, then try hyperparameter tuning using different methods depending on the model:

- Random Search
- Grid Search
- Manual Comparison

Notes: We decided to favour Random Search over Grid Search for some more complex models, such as Random Forest, and that would take too much time to run with Grid Search.

After the hyperparameter tuning, we noticed a great improvement in all the models, but the ones that showed the best results are summarized below:

Model	Parameters	Val F1_score before hyperparameter Tuning	Val F1_score after Hyperparameter Tuning
Gradient Boost Classifier	n_estimators =80, subsample = 0.8, max_features = 0.5, min_samples_split=1000, min_samples_leaf= 70, random_state = 15	66.20%	68.18%
Random Forest	n_estimators = 670, criterion = 'entropy', min_samples_split= 5, min_samples_leaf= 4, max_features= 'sqrt', max_depth= 80, bootstrap= True, random_state = 42	59.18%	66.18%
Support Vector Machine	kernel='rbf', C=10, gamma=0.01	39.46%	65.54%

3.2 Model Assessment:

After applying predictive models and using hyperparameter tuning, we need to assess each model's performance. In this step since we can not rely on the accuracy, we decided to use different approaches: Confusion matrix, classification report, and the f1_score.

We will compare the different results for the best two performing models:

- **Gradient Boost Classifier**

Given that this is an imbalanced dataset, the GB classifier, like most models, managed to predict the majority label better. Out of all the non_buyers, 95.50% were predicted correctly. While out of the buyers, only 64.37% were predicted correctly.

- **Random Forest**

The random forest did a better job in predicting the non-buyers than the GB boost. Out of all the non-buyers, 97.23% were predicted correctly. However, out of the buyers, only 55.19% were predicted correctly. This explains why the random forest had a worse recall of 56% compared to 64% in GB.

The results for the GB model are represented in Figure 4 (Annexes) and in Figure 5 (Annexes) for the results of the Random Forest model.

Finally, to assess the performance of all the models together, we built a ROC curve (Figure 8 - Annexes). According to our ROC curve, the best classifier is AdaBoost Classifier with the highest

AUC of 0.82, followed by Gradient boost and decision tree with an AUC of 0.8. The random Forest showed some good results as well, with an AUC of 0.78.

4. Let's discuss the Results

Looking at the performance of our models, we could conclude that Random Forest and Gradient Boost Classifier seem to give us the best scores, a little above 0.68 for GB and around 0.66 for RF. Therefore, we tried to combine the outputs of those two models and to run them as a stacking model. With this ensemble method, we were expected to boost our predictive accuracy, but after running the new model, it happened to have a lower score. We can observe this by comparing the results of the initial score (Figure 6 - Annexes) with the ones from the stacking model (Figure 7 - Annexes). We, therefore, decided to drop the idea and keep the models as they are.

Even though we identified our best-performing models, we decided to predict the test dataset labels using all the models we worked with just so we can have a better comparison.
Below we summarized the results:

Model	Train Score	Validation Score	Test Score
KNN	67%	64%	66.6%
LG	56.68%	61.65%	63.41%
SVM	63.71%	65.54%	68.60%
Decision Tree	65.22%	61.20%	66.99%
Neural Networks	66.44%	65.59%	65.53%
Random Forest	81.16%	66.18%	72.18%
GB Classifier	68.41%	68.18%	66.66%
AdaBoost	66.53%	64.66%	65.74%
Stacking	78.49%	66.97%	69.66%

5. Discussion:

During this project, we got the chance to apply our theoretical knowledge learned throughout this semester into a practical, real case project. One thing we got to learn while working on this assignment, is that there is no general rule applicable to every case. Every model is different, requires different approaches, and can generate different results. The theoretical concepts learned in class are not always applicable and can give unexpected and often hard to explain results. However, that is the beauty of the practical cases, making sense of the results achieved, which is something we tried to understand and work on throughout this project. One example is the fact that the methods used to deal with the imbalanced dataset did not work, or the fact that the feature selection methods gave contradicting results and did not improve the models' scores. All of these unanticipated results made us curious to learn, read and search more about these concepts.

6. CONCLUSION

After working on different prediction models, we would certainly recommend TechScape to go with a tree-based model because they were the ones with the best results. The random forest prediction model was the best out of all the models used. However, with this model specifically, we can notice a little overfitting problem, the model performed 9% better on the train than on the test dataset and while this is understandable as most models perform better on the training dataset, trying to reduce the difference has worsened the model score even more. This issue leaves room for improvement in this project, along with other matters that were not properly addressed. For instance, the issue of unbalanced data was not fully solved, as we could not implement a technique that made a positive effect. This point can be considered for future research.

A final point to be addressed is the fact that the company had a break in sales in April and the pandemic might have changed the way people consumed the product, so this could have also influenced and made it harder for the algorithm to predict, making also a valid point for future considerations.

REFERENCES

Alpaydin, E. (2014). Introduction to Machine Learning.

Biau, G., & Scornet, E. (2016). A Random Forest Guided Tour.

Breiman, L. (2001). Random Forests.

Cawley, G. C., & Talbot, N. L. (2010). On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation.

Kelleher, J.D., Namee, B.M., D'arcy, A. (2015). Fundamentals of Machine Learning for Predictive Data Analytics.

Lantz, B. (2013). Machine Learning with R.

Tallón-Ballesteros, A. J., & Riquelme, J. C. (2014). Deleting or Keeping Outliers for Classifier Training?

Wirth, R., & Hipp, J. (2000). CRISP-DM: Towards a Standard Process Model for Data Mining.

Brownlee, J. (2021). accessed 2 December 2021, https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Satpathy, S. (2020), accessed 2 December 2021, https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/

Brownlee, J. 2021, accessed 2 December 2021, https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning, Guillaume Lemaitre, et al., JMLR 18, pp. 1-5, 2017

Brownlee, J. 2020, accessed 8 December 2021, https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/

Idil Ismiguzel 2021, accessed 8 December 2021, https://towardsdatascience.com/hyperparameter-tuning-with-grid-search-and-random-search-6e1b5e175144

Hrouda-Rasmussen, S. accessed 19 December 2021, https://towardsdatascience.com/quadratic-discriminant-analysis-ae55d8a8148a

Scikit-learn: Machine Learning in Python, Pedregosa, et al., JMLR 12, pp. 2825-2830, 2011

Matthias Döring 2018, accessed 19 December 2021, https://www.datascienceblog.net/post/machine-learning/linear-discriminant-analysis/

Annexes

1. Creativity and Other Self-Study

A Hyperparameter is a parameter of a model whose value is provided by the user; it's a parameter that the model can't estimate from the training data.

Hyperparameter Tuning is a technique used to tinker with the values of the hyperparameters, in an attempt to find the value combinations that will lead to the best model scores. The tuning methods used in this assignment were Grid Search and Random Search.

In **Grid Search**, the user starts by specifying the range of values for each hyperparameter in the model. All possible combinations of hyperparameter values will form a grid. This method will calculate the score for each combination, and the user will then be able to use that combination of values for their model. This method can take a long time to compute, depending on the size of the grid or the specified values themselves.

In **Random Search**, the user once again starts by specifying the range of values for each hyperparameter in the model. However, not every combination of values will be tested; instead, the user will provide several iterations, and in each iteration, the method will randomly select a combination of values and calculate the respective score. This method can take less time to compute depending on the number of iterations and is a useful way to test hyperparameter value combinations that are very different from each other (or possibly even unintuitive), in order to be pointed in the right direction faster.

Quadratic Discriminant Analysis (QDA) is a method that helps the user identify which variable combinations are best at predicting to which cluster certain data points will belong to. In other words (and like the name suggests), this analysis helps the user discover if there is a combination of variables/features that *discriminate* between two or more groups/clusters/classes. Unlike Linear Discriminant Analysis (LDA), this method is capable of non-linear data discrimination.

Synthetic Minority Oversampling Technique (SMOTE) is a method used to add some balance to an unbalanced dataset. When the majority class has too many data points compared to the minority class, the results of the classifier model can be biased.
To avoid this, the SMOTE method is used; it creates new data points with similar characteristics to the data points in the minority class. It does this by:
1) taking a randomly selected data point from said minority class;
2) looking for k of its nearest neighbors (by default, it's usually k = 5, but this value can be changed by the user);
3) selecting one of these k neighbors;
4) generating a new data point located between the data points of steps 1) and 3).

Simply duplicating data points would help with the imbalance, but would provide the model with no new information.

Random Resampling Techniques
The following techniques are also used with the intent of balancing unbalanced datasets.

Random Oversampling is a method in which data points from the minority class are randomly selected (with replacement), duplicated, and then added to the training dataset. One limitation of this method is the possibility of overfitting from having too many duplicates

data points.

Random Undersampling consists of randomly taking data points from the majority class and deleting them from the training dataset.
One problem with this method is that important data points could end up being randomly selected and deleted, thus affecting the fit of the model and its overall performance.

These resampling techniques are only applied to the training data, and not the test data. The objective is to influence the fit of the model so that it can adjust itself to all sorts of new data.

It is possible to combine both these Random Resampling methods, or even combine SMOTE with Random Undersampling.

2. Figures

Figure 1: Redundancy correlation matrix

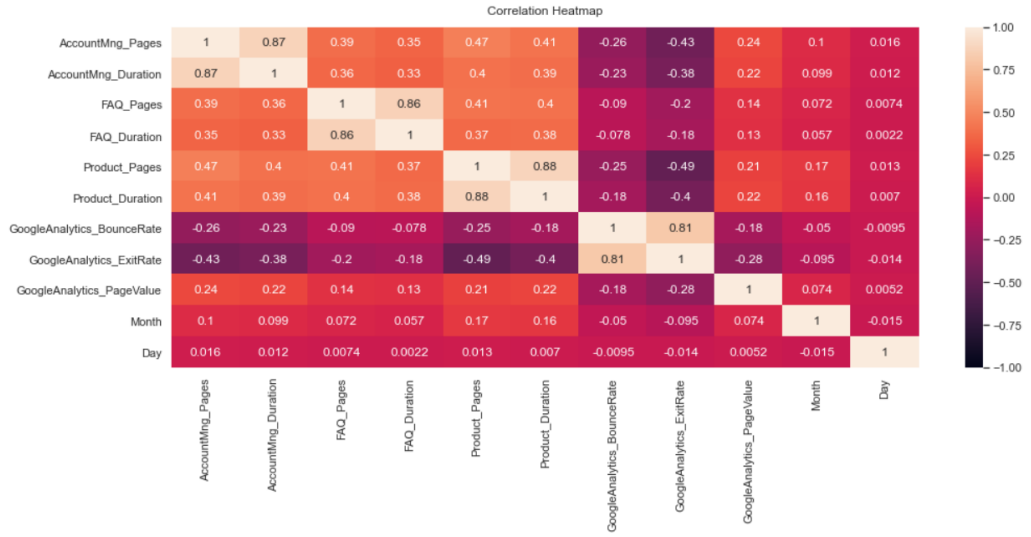

Figure 2: Table of the mean of the Metric Features depending on the label of the target variable.

Buy	AccountMng_Pages	AccountMng_Duration	FAQ_Pages	FAQ_Duration	Product_Pages	Product_Duration	GoogleAnalytics_BounceRate
0	2.129158	74.382501	0.457322	32.039180	28.745827	1086.765031	0.025475
1	3.385954	121.778168	0.784149	59.092305	47.687500	1867.399982	0.005053

Figure 3: Summary of the Feature selection's results, per method

Numerical Data

Predictor	Spearman	RFE	Lasso	Decsion Tree	What to do? (One possible way to "solve")
AccountMng_Pages	Discard	Discard	Keep	Keep	Try with and without
AccountMng_Duration	Discard	Discard	Discard	Keep	Discard
FAQ_Pages	Discard	Discard	Discard	Keep	Discard
FAQ_Duration	Discard	Discard	Keep	Keep	Try with and without
Product_Pages	Discard	Discard	Keep	Keep	Try with and without
Product_Duration	Discard	Discard	Keep	Keep	Try with and without
GoogleAnalytics_BounceRate	Discard	Keep	Keep	Keep	Keep
GoogleAnalytics_ExitRate	Discard	Keep	Keep	Keep	Keep
GoogleAnalytics_PageValue	Keep	Keep	Keep	Keep	Keep
Month	Discard	Keep	Keep	Keep	Keep
Day	Discard	Discard	Discard	Keep	Discard

Categorical Data

Predictor	Chi-Square
OS	Keep
Browser	Discard
Country	Discard
Type_of_Traffic	Keep
Type_of_Visitor	Keep

Figure 4: GB Classifier Confusion Matrix

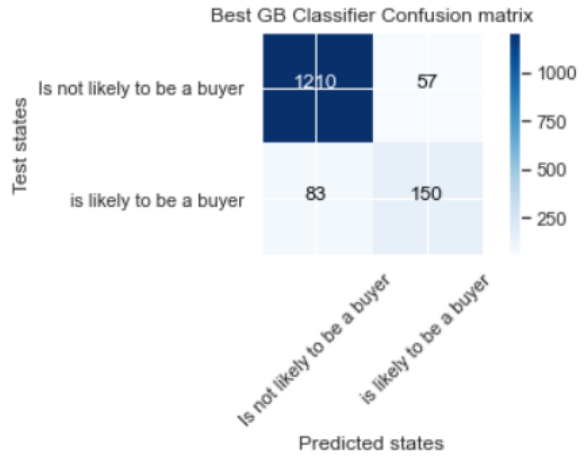

Figure 5: Random Forest Confusion Matrix

Figure 6: Bar chart of the F1 scores of the models without the stacking model.

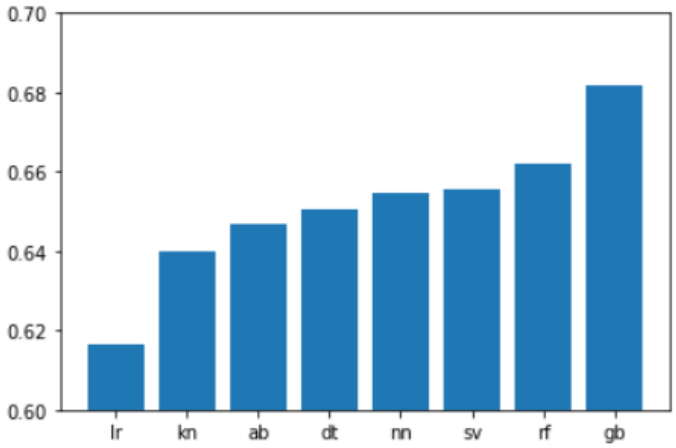

Figure 7: Bar chart of the F1 scores of the models with the stacking model

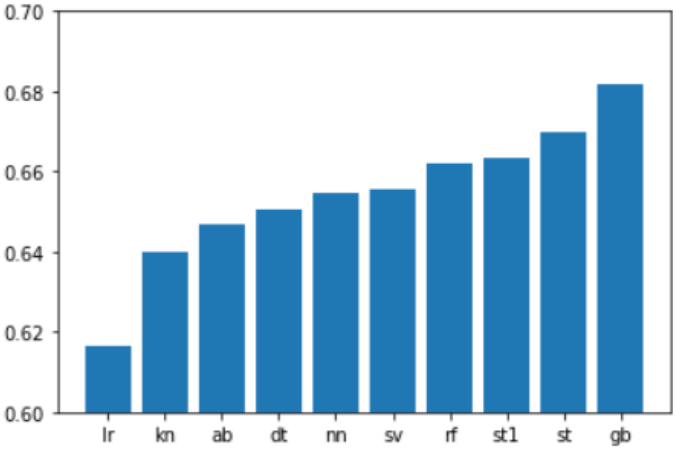

Figure 8: ROC Curve