Project Report: Group 36
Débora A. Santos (m20200748), Eleonora Sbrissa (m20200628),
Luís Reis (m20200636), Pedro Godeiro (m20200396), Rupesh Baradi (m20200994)

ABSTRACT

As part of a Machine Learning Project, this paper has the goal to implement Machine Learning techniques to create a Learning Algorithm used to classify two different groups into an artificial dataset created for the sole purpose of this project. The dataset for this study was provided by the Newland government and it included information related to 32.500 actual residents with 15 attributes. Since the government of Newland uses a tax system in which people with income below average pay a different percentage of tax than people with income above average the main goal of this project was to develop an algorithm to predict the income levels of new residents coming from Earth. Having that in mind, on this paper we will discuss some techniques used for data pre-processing such as RFE, Decision Trees, Outliers Removal, KNN Imputer and Oversampling. For the creation of the final model different approaches were used as Logistic Regression, Neural Networks, Random Forest, Gradient Boosting and Bagging techniques. After all that we will show how we decided which ones are the best techniques to be used on our Predictive Model for this specific problem, evaluating the performance of the models and analyzing which characteristics of the population have more influence on our classifier.

Keywords: Project, Machine Learning, Gradient Boosting, Predictive Models, Data pre-processing.

1. INTRODUCTION

Considering the hypothetical problem of this project, we have a situation in which a new planet was discovered and started being populated. Newland has a binary tax system in which people who have an income below average pay 15% of their income, while people above average contribute with 30% of their yearly income. In 2048, there are multiple ships coming to start living in Newland and the government wants to predict if the future new residents are going to have their income above or below average, so they can actually know how much people are going to pay on taxes. Having said that, the government is looking for a group of Data Scientists who can create the best model to predict that.

The main objective of this work consists in implementing machine learning techniques and algorithms to predict the class of income of these new residents. This type of problem is called classification. *The output to be predicted is the actual or the probability of an event/class to be predicted can be two or more.[1]*

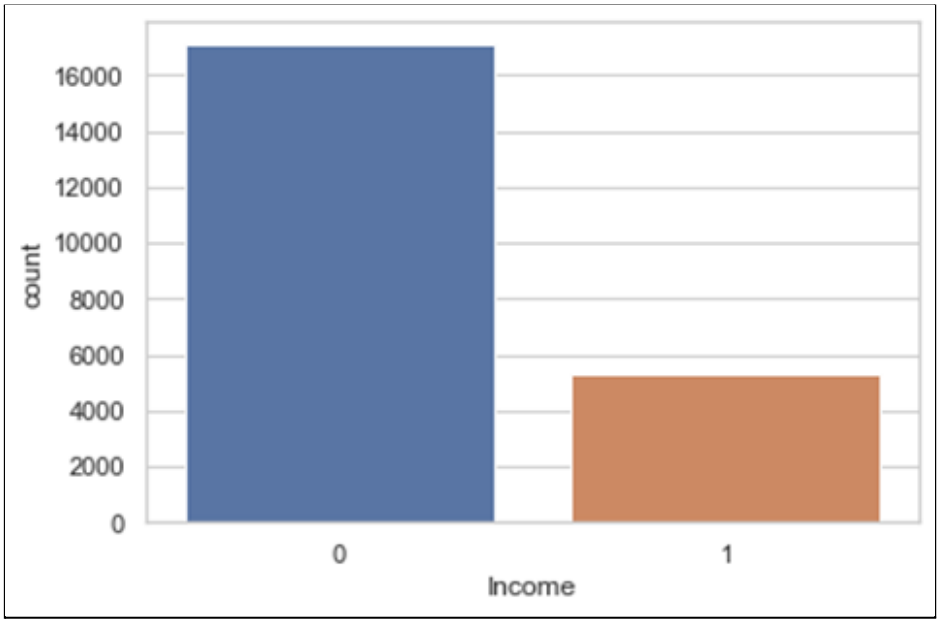
The data for this study was provided by the Newland government and it included information related to 32.500 residents of Newland in 2047. Some descriptive information of the residents is included in all the observations, but we have knowledge of the class of income only for 22400 people. As the class of each resident was provided, even so only in a part of it, this problem is considered as supervised learning.

Another important objective of this study is to make a good descriptive analysis of the past data. The process of data understanding, and data preparation are very important to get a high performance of the final model. According to Swamynathan, "*It is important to remember that the foundation of predictive analytics is based on probabilities, and the quality of prediction by statistical algorithms depends a lot on the quality of input data".[1]*

2. BACKGROUND

One of the problems that we may face in a classification problem is dealing with unbalanced data. Most models assume that the data has a good distribution between the classes. In this study, the data provided was unbalanced: there were more observations with income below the average, as we can see in Figure 2-1.

Figure 2-1: Distribution of Income Labels



This type of issue is known as imbalanced classification. The main techniques used to solve this problem are oversampling, under sampling, threshold moving, and ensemble techniques.

In this study, we used the Synthetic Minority Oversampling Technique (SMOTE) to balance the target variable. This approach involves duplicating examples in the minority class, although these examples do not add any new information to the model [2]. It changes the training data distribution in order to the rare class to be well represented.

SMOTE is part of the imblearn package in python. We decided to keep the standard parameters for our analysis. [3] Using SMOTE was fundamental to compare different classifiers used in this project, since we applied it to balance the target variable.

3. METHODOLOGY

The dataset provided by the government is divided into 2 subsets: 22.400 observations were used to develop and train the predictive model, and the remaining were used by the government to test the quality of the model.

The programming language used to access the data and develop the model was Python. In this study the main libraries used were pandas to explore and transform the data, matplotlib and seaborn for visualizations, imblearn to deal with class imbalance and sklearn to explore machine learning models.

The main steps of this study were: exploration of the data, transformation, feature selection, creation of different models, evaluation and choice of the one with best performance to be uploaded on Kaggle.

The first step consisted in an Exploratory Data Analysis. '*EDA' is all about understanding your data by employing, summarizing and visualizing techniques. At a high level, EDA can be performed in two ways: univariate analysis and multivariate analysis.[1].*

For our analysis we chose to investigate data through univariate descriptive statistics. We checked the data type of each feature and the summary of the central tendency for each one of them. Some visualizations were created as bar plots, box plots and histograms to see how the data was distributed. We also did a correlation matrix to understand if there was any relationship between the variables.

The exploratory analysis was good to understand the presence of missing values, outliers, irregular cardinality, and the relationship of the variables.

On the feature engineering step, we created and deleted variables, imputed missing values and outliers were removed using manual filters. The Table 3-1 shows a summary of feature transformation.

Table 3-1: Summary of Feature Engineering

Variable	Feature Transformation	Comment
Age	Create variable	Created from Date of Birth
Gender	Create variable	Created from the presence of "Mr." on Name
Type of Traveler	Create variable	A=Traveler did not received money neither paid to move to Newland B=Traveler received money to move C=Traveler paid to move
Date of Birth	Remove variable	Not useful as we now have Age
Name	Remove variable	Not useful as we now have Gender
Citizen ID	Put as index	Useful only as Index
Base Area	Replace missing values	It has missing values
Employment Sector	Replace missing values	It has missing values
Role	Replace missing values	It has missing values
Money Received	Remove outliers	We considered only Money Received < 120.000
Years of Education	Remove outliers	We considered only 4<= Years of Education <= 20

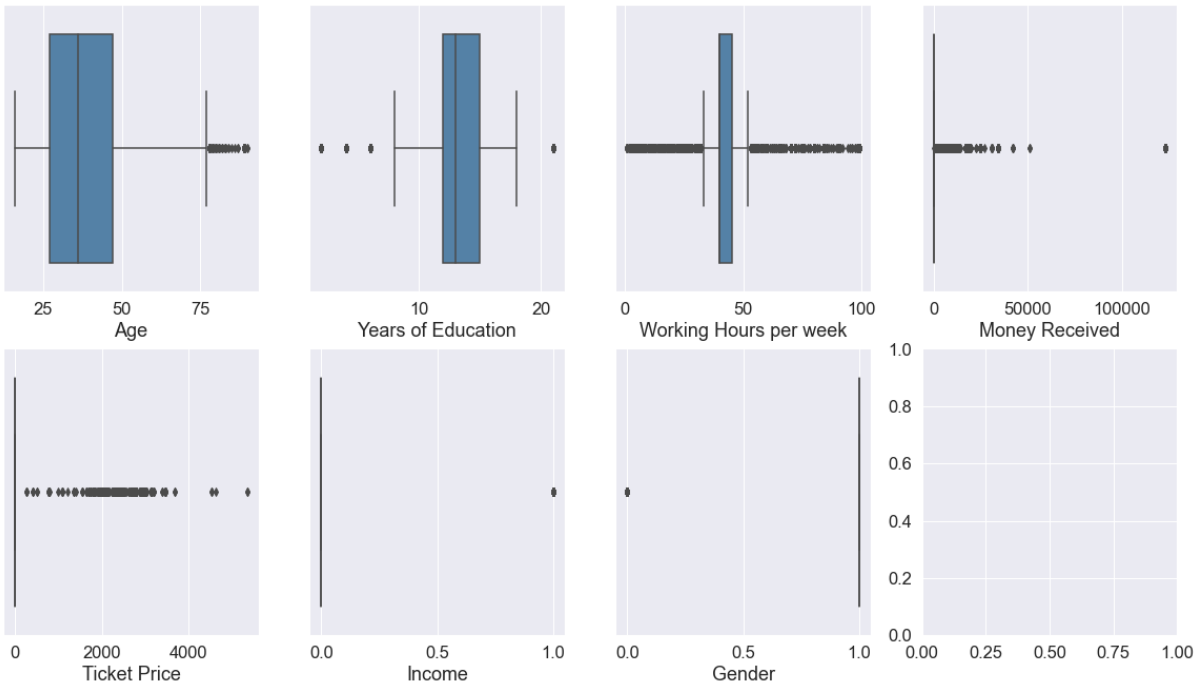
Regarding the creation of variables, we realized that we could create more meaningful information out of the data we already had. One of the variables we created was Age, in which we used the date of 30/06/2048 as reference date to transform Date of Birth into Age, since we are in the year of 2048 and using 30/06 is a correct assumption by demographic literature as a Mid-year Population [15]. As a consequence, we exclude DOB variable. Gender was another variable created, in which if we could see the presence of the string "Mr." on variable Name, we could consider it as a man and otherwise as a woman. Considering that, we remove the Name variable. The last variable we created was Type of Traveler, in which if we have any register on variable "Money paid" it would be Group C, if there is a register in "Money received it is Group B and if we don't have it in either, it is Group A.

Taking into consideration the Citizen ID, we realized there would be no point in using the Citizen ID feature as a value who could predict the target variable, since there is no logical correlation with that, but we decided to use it as Index of our dataset so we could locate better all our information.

Regarding missing values, we have three categorical features to impute. The easiest way is to just impute the missing values as the mode, but that's not the only way. Another approach is to use KNN Imputer [6], which is a more sophisticated way to impute values. The problem with using KNN Imputer is that we need to have numeric data, so we need to encode the data and then do the KNN Imputation. We did that, and comparing the results with Mode Imputation, we got better results with the last one. By consequence, we decided to impute the missing values as the same value of the mode of each feature.

For the outlier removal, we tried to apply the IQR method, but we would lose more than 50% of the data. There are many observations out of quantiles that they are really close to each other, as you can see in Figure 3-1. We used some reasoning for each variable and applying manual filters on variables Money Received and Years of Education, we remain with 98.1% of our database.

Figure 3-1: Boxplot Distribution of Numeric Variables



In order to prepare the data to do the feature selection, the next step of feature engineering was to transform the data in binary variables, since we were dealing with some categorical features. Having that in mind, we used One Hot Encoding on all categorical features that changed our dataset to only metric features. We also needed to standardize our data to do Feature Selection. We could use many different scalers for that, like MinMax, Standard or Robust. Considering that Robust Scaler is robust to marginal values and had a better performance on our model, we decided to go with Robust. *RobustScaler is based on percentiles and therefore not influenced by a few numbers of very large marginal outliers*.[14]

On the feature selection, in order to find the optimal number of features, we decided to apply a combination of two methods: Recursive Feature Elimination (RFE) and Decision Tree Classifier (DT). Tables 3-2 and 3-3 show the parameters for both methods.

Table 3-2: Parameters chosen for RFE

RFE	
Parameter	**Values**
Estimator	Random Forest
Cross validation	Stratified KFold
Step	1
Scoring	F1_Micro

Table 3-3: Parameters chosen for Decision Tree

Decision tree	
Parameter	**Values**
Tree depth	12
Criterion	Entropy

On the next step, having done all of the Data Pre-processing and Feature Selection, our Data was ready to be used to create a Machine Learning Model. We used a number of algorithms in our Data, such as: Logistic Regression, Random Forest, AdaBoost, Gradient Boosting and Stacking.

Having the model and just applying it into the Data is not enough. Another important step of modeling is to define the parameters of each classifier. In our project, we first tried to apply the models with default parameters, and then we started tuning the hyperparameters using our validation dataset.

We decided to tune the hyperparameters by using the Grid Search method in the model with the highest score to calculate the best parameters. In addition, we tried to combine all of them by stacking different classifiers into a more complex classifier.

In order to measure the performance of our models, we used a 10 fold cross validation by creating an algorithm to calculate the average score of each fold, which was calculated by the F1 score 'micro', the same score that is going to be considered by the Newland Government Entity to choose the best model.

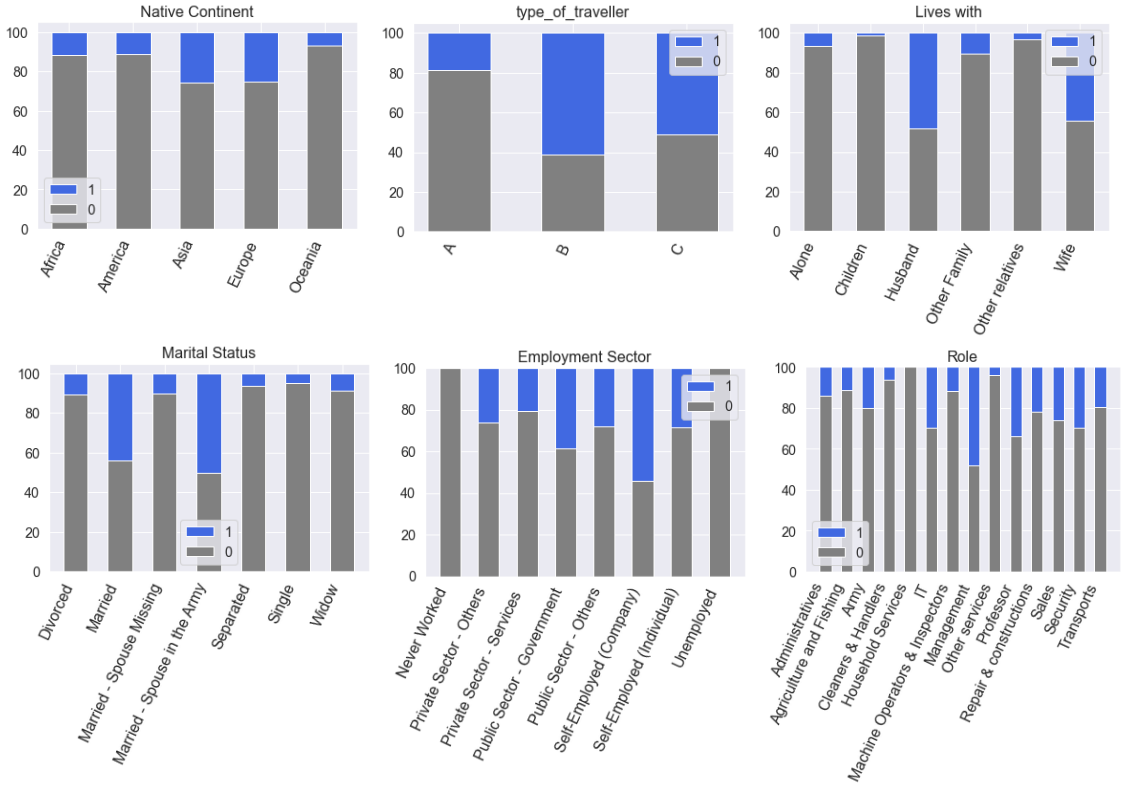
And finally, we needed to apply all of the transformations done on the Training Data on the Test Data, predict the class with our best model and submit results to Kaggle. As expected, considering the way the Kaggle platform evaluates models in the public leaderboard, we had different results between Kaggle and our Cross Validation Score. Even so, we decided that our best model on Cross Validation would be the final one because it is a more reliable method of testing a Machine Learning Model.

4. RESULTS

4.1 Data Exploration

First, we would like to have a look at the income distribution in the categorical variables. Even though we have an imbalanced dataset, with just 22% of people with income above the average, we can see that it's not equally distributed among the different types of answers for each categorical feature, as we can see in Figure 4-1.

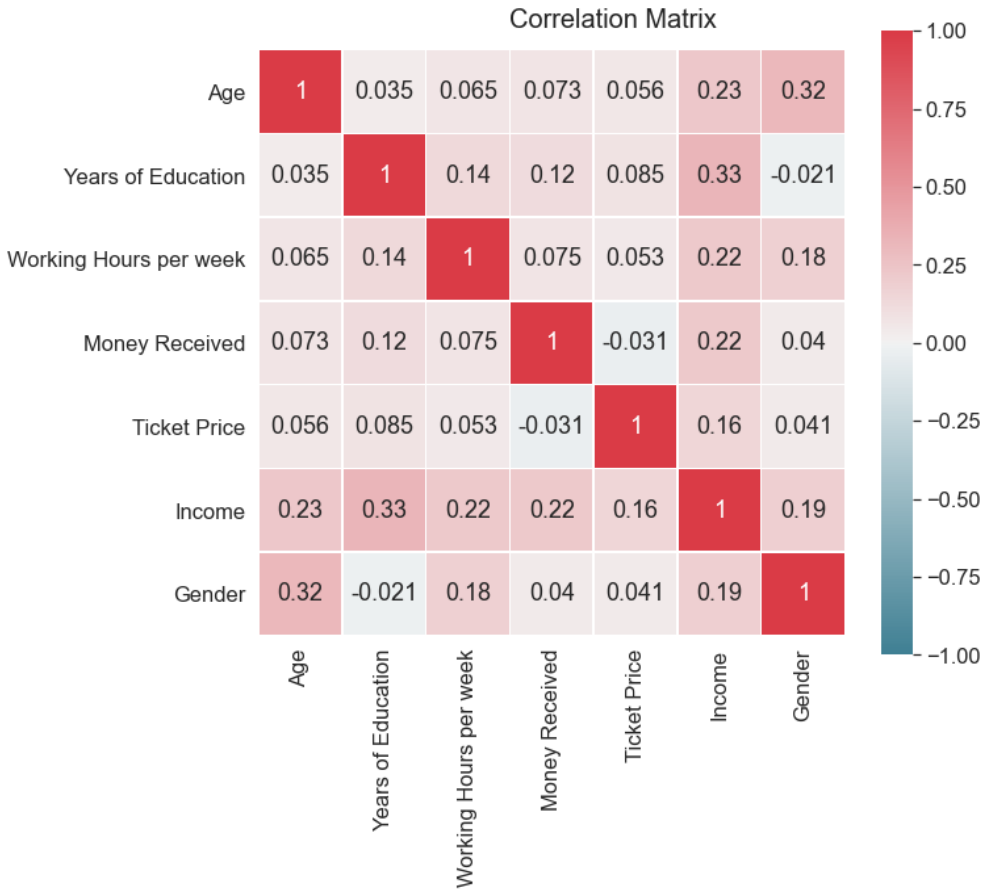
Figure 4-1: Barplot of Income Distribution by Categorical Variables



As we analyze Figure 4-1, we can see that the categorical variable was able to give us some interesting knowledge about our target variable, something that is going to be valuable on our final model. We were able to find that some variable characteristics may have a bigger presence of people with income higher than average than other variable characteristics in the same feature. As examples of characteristics of people with high income we have: Being from Asia or Europe, being or having paid to be in Newland, living with your spouse, being married and some specific Employment Sectors and Roles.

Another interesting thing that needed to be analyzed was the correlation of the metric features with our target variable, Income. The correlogram for that can be visualized in Figure 4-2.

Figure 4-2: Correlogram of Metric Features and Target Variable

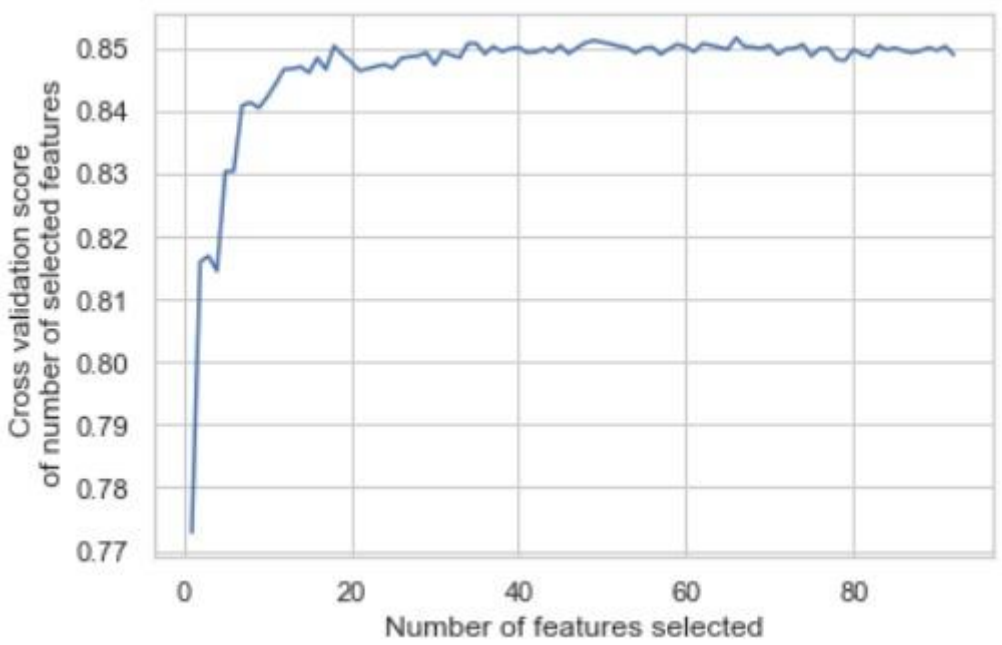


When we analyzed Figure 4-2, we could see that there isn't a big correlation between our target variable and the metric features, what is expected, but they all have some kind of small correlations which together can become something more powerful to our model.

4.2 Feature Selection

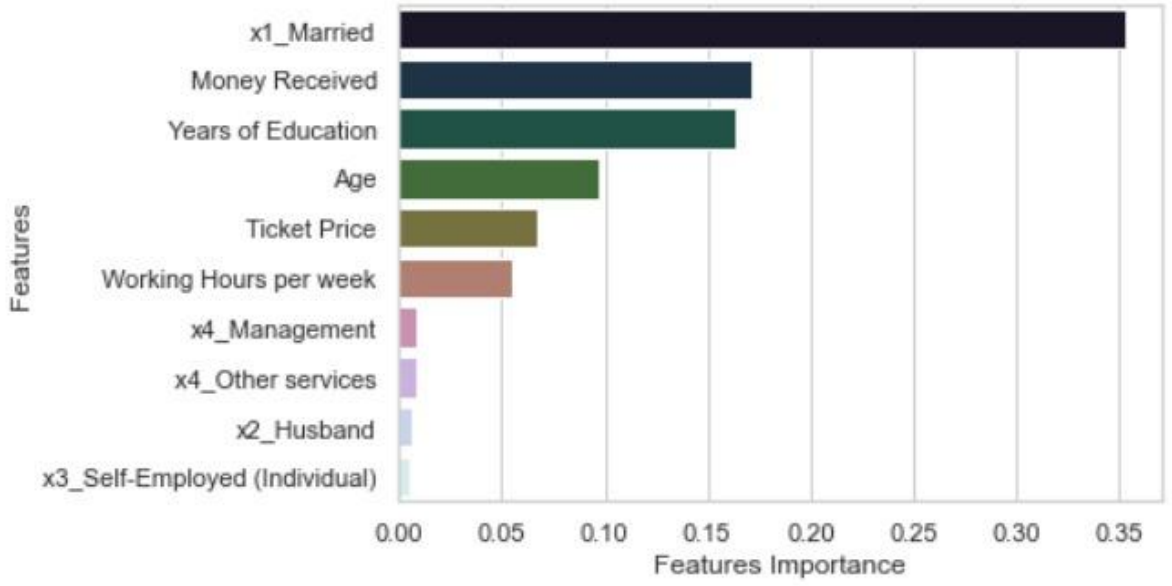
As said before, we used a combination of two different Feature Selection methods to choose which features we would use in our model. The first one we used was the RFE,[4] in which we had a Random Forest Classifier as our classifier. After using RFE, the algorithm chose 66 out of the 92 features as the optimal number of features, as we can see on figure 4-3.

Figure 4-3: RFE score by number of features



The other algorithm used for feature selection was a Decision Tree Classifier. This method classifies the features according to their importance. From the 92 features, we kept 57, which are the ones with a feature importance greater than 0. The figure 4-4 shows the Top 10 features on feature importance.

Figure 4-4: Importance of Features by Decision Tree Classifier



So after evaluating both methods, we decided to use on our predictive models only the 56 features that were selected both by the RFE and Decision Tree.

4.3 Model Selection

We explored some different classifiers to compare in order to get the best model possible. The first criteria used to select the best model was the F1 Score Micro, as it was the one chosen by the Newland Government. The second one was the false positive ratio since it can be a problem for the government if our algorithm predicts more people paying 30% of taxes than what they pay in reality, causing a deficit on taxes for the government.

What follows is an explanation of the difference between the raw models (classifiers with default parameters), and the ones we created by tuning the parameters. The comparison between the default classifiers is represented on Table 4-1.

Table 4-1: Comparison between default classifiers

	Time	Train	Test
LogRegression	0.448+/-0.04	0.829+/-0.01	0.829+/-0.01
RandomForest	4.037+/-0.55	0.981+/-0.0	0.85+/-0.01
AdaBoost	1.627+/-0.28	0.861+/-0.0	0.859+/-0.01
GradientBoost	3.745+/-0.49	0.869+/-0.0	0.864+/-0.01

When we analyzed the results in Table 4-1, we could see that Logistic Regression Classifier did not fit well to our validation data, Random Forest Classifier is hugely overfitting and AdaBoost and Gradient Boosting Classifiers were the only ones which had stable performances and good performances on our Validation set, but Gradient Boosting was the one who is performing better between both of them. Now, after comparing the performance of each default model, we are going to go deeper into the analysis of each one of them.

4.3.1 Logistic Regression Classifier

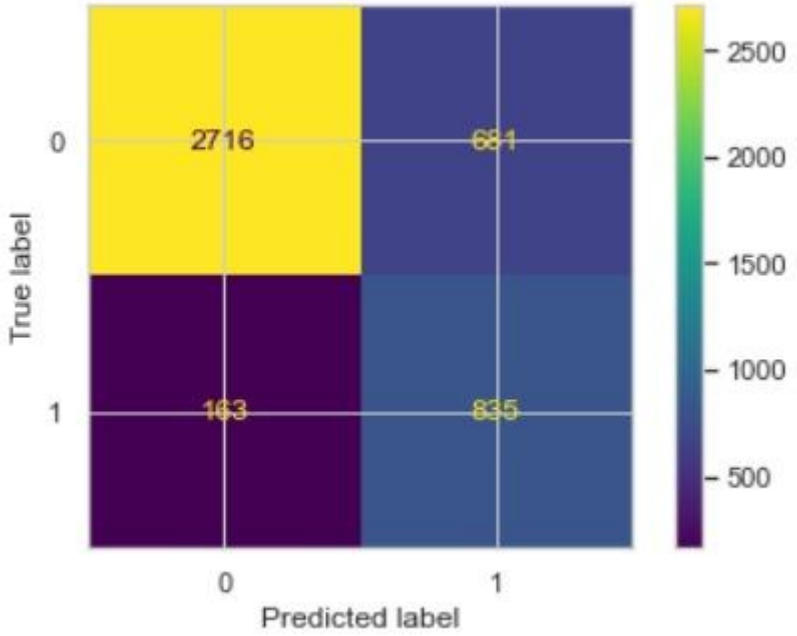
After doing parameter tuning on this classifier,[8] we were able to have the results shown on Table 4-2.

Table 4-2: Performance of the Parameter Tuned Logistic Regression Classifier

	Time	Train	Test
LogRegression	3.495+/-0.11	0.852+/-0.0	0.85+/-0.01

And the Confusion Matrix for that classifier is presented on Figure 4-5.

Figure 4-5: Confusion Matrix of the Parameter Tuned Logistic Regression Classifier



In general, we could conclude that the model is performing reasonably well . It's not overfitting, but we couldn't confirm that early in the project that this was the best model we could find. Having that in mind, we continued exploring other classifiers.

4.3.2 Random Forest Classifier

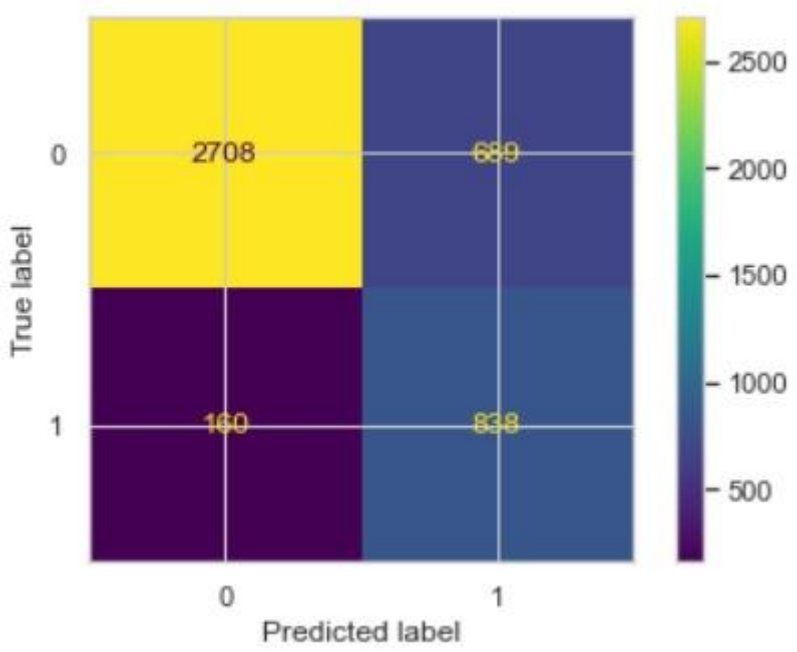
After doing parameter tuning on this classifier,[7] we were able to have the results he can see on Table 4-3.

Table 4-3: Performance of the Parameter Tuned Random Forest Classifier

	Time	Train	Test
RandomForest	2.143+/-0.19	0.865+/-0.0	0.858+/-0.01

And the Confusion Matrix for that classifier is presented on Figure 4-6.

Figure 4-6: Confusion Matrix of the Parameter Tuned Random Forest Classifier



After the parameter tuning, we could reduce by much the overfitting of the Random Forest Classifier and improve its result on our Validation set. When we compare with the Logistic Regression, we see that Random Forest is slightly better, but we kept exploring different classifiers.

4.3.3 AdaBoost Classifier

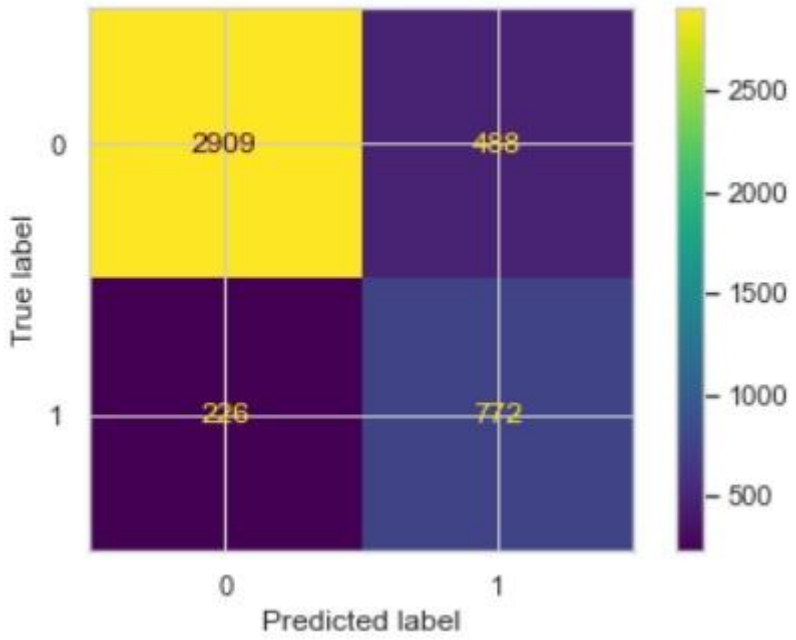
After doing parameter tuning on this classifier,[10] we were able to have the results he can see on Table 4-4.

Table 4-4: Performance of the Parameter Tuned AdaBoost Classifier

	Time	Train	Test
AdaBoost	2.829+/-0.23	0.867+/-0.0	0.866+/-0.01

And the Confusion Matrix for that classifier is presented on Figure 4-7.

Figure 4-7: Confusion Matrix of the Parameter Tuned AdaBoost Classifier



And analyzing AdaBoost we could see that we had a much better model on Testing Score, with overfitting close to zero and reduced a lot the number of false positives that we got. Having that we mind, we could consider that the Parameter Tuned AdaBoost was a really good classifier, but we still wanted to keep checking for different classifiers.

4.3.4 Gradient Boosting Classifier

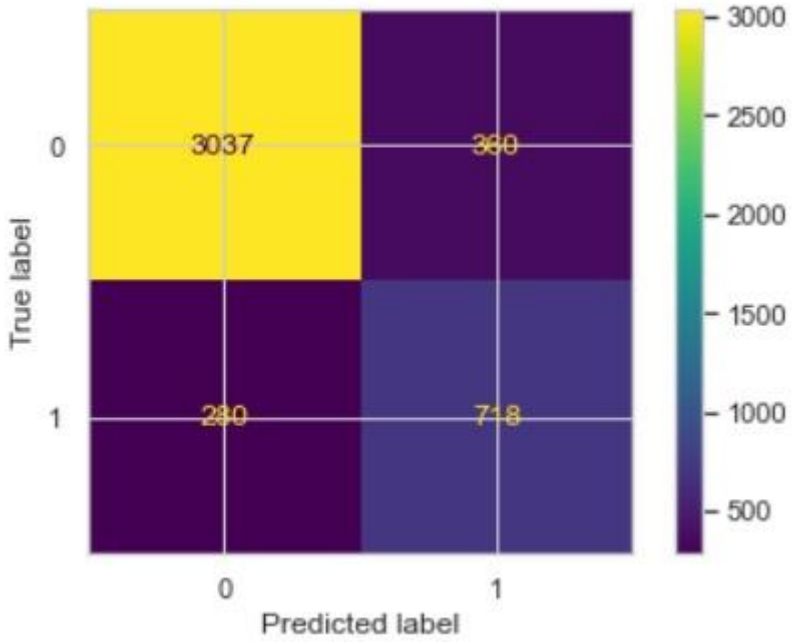
After doing parameter tuning on this classifier,[9] we were able to have the results he can see on Table 4-5.

Table 4-5: Performance of the Parameter Tuned Gradient Boosting Classifier

	Time	Train	Test
GradientBoost	7.428+/-0.26	0.888+/-0.0	0.874+/-0.01

And the Confusion Matrix for that classifier is presented on Figure 4-8.

Figure 4-8: Confusion Matrix of the Parameter Tuned Gradient Boosting Classifier



And looking to Table 4-5 and Figure 4-8, we could see that our Gradient Boosting Classifier, even though having a slight overfitting, was the best classifier in terms of Testing Score, being better than the AdaBoost by a significative margin and being the best model on avoiding false positives, which is something we want to avoid as much as possible. Having this as the best Classifier we could find, we still wanted to try one last classifier.

4.3.5 Stacking Classifier

After trying some different classifiers, we tried to combine them using a Stacking Classifier[11] with our parameters tuned Logistic Regression, Random

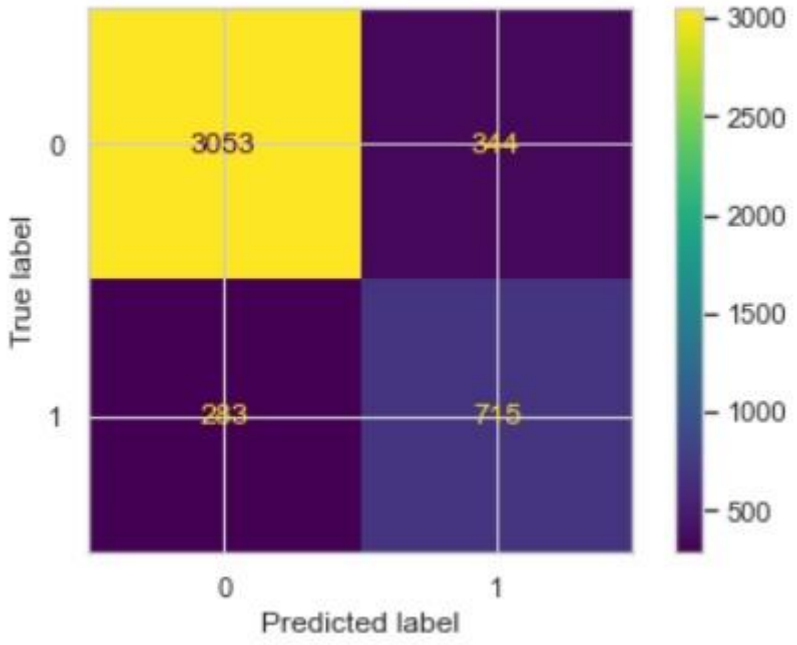
Forest, AdaBoost and Gradient Boosting as base learners and a Logistic Regression as a Meta Learner, and the final performance is represented as on Table 4-6.

Table 4-6: Performance of Stacking Classifier

	Time	Train	Test
Stacking	53.86+/-3.31	0.878+/-0.0	0.869+/-0.01

And the Confusion Matrix for that classifier is presented on Figure 4-9.

Figure 4-9: Confusion Matrix of the Stacking Classifier

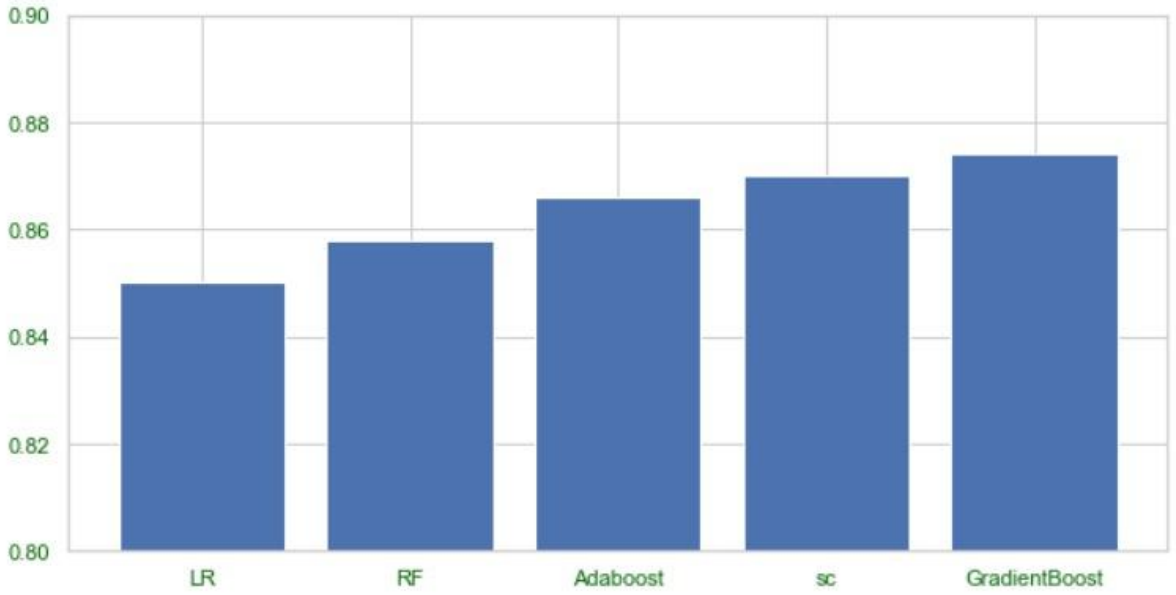


And as we can see on Table 4-6 and Figure 4-9, even though we could reduce the number of false positives, the score of our model turned out to be lower on stacking when compared to Gradient Boosting, which can be an evidence that our Gradient Boosting Classifier is good enough for the problem being studied.

4.3.6 Final Classifier

After doing all of the analysis of different classifiers, we can compare their performance on the Figure 4-10.

Figure 4-10: Comparison of the different Classifiers



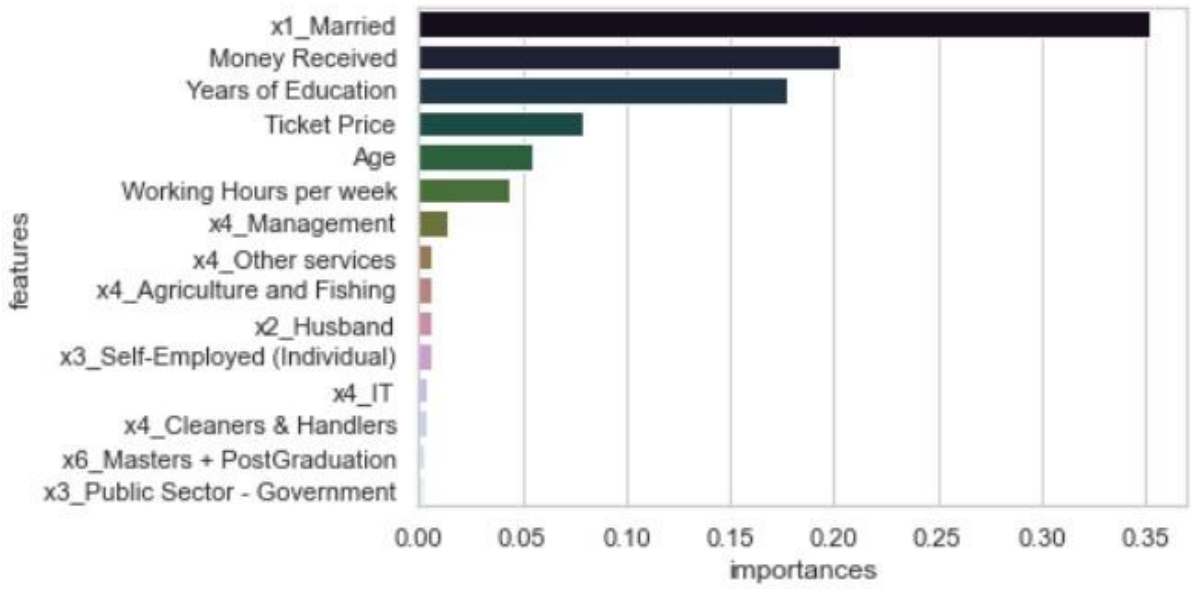
After analyzing all the Tables from 4-1 to 4-6 and Figures from 4-6 to 4-9, having as support Figure 4-10, we came to the conclusion that our Parameter Tuned Gradient Boosting Classifier was the best Classifier we could find for the problem of this study. Even though we had a lower number of false positives in our Stacking Classifier, we considered that having a worse F1 Micro Score and a much more complex classifier would not be worth it to reduce the false positive rate slightly.

For our final Gradient Boosting Classifier, the chosen parameters were a learning rate of 0.2, maximum depth of 9, number of estimators of 90 and a minimum sample split of 1100.[5]

And talking about the performance of the model, from the distribution of data we can calculate an overall accuracy rate of 85.44%. And as mentioned in section 4.3, considering the nature of our problem, we need to avoid misclassifying people as higher than average income, and our model had a result in which only 8.4% of the people were incorrectly identified as high income. And when it comes to predicting people with higher than average income, we identified that 33.3% were misclassified as lower than average, the best result for all of our classifiers. All of these results can be checked on Figure 4-8.

Regarding the features importance, we can see on Figure 4-11 the Top 15 features using Gradient Boosting.

Figure 4-11: Feature Importance of Gradient Boost



On Figure 4-11 can see that the top 7 features are the same identified by the Decision Tree Classifier, and they are accountable for 88% of the feature importance.

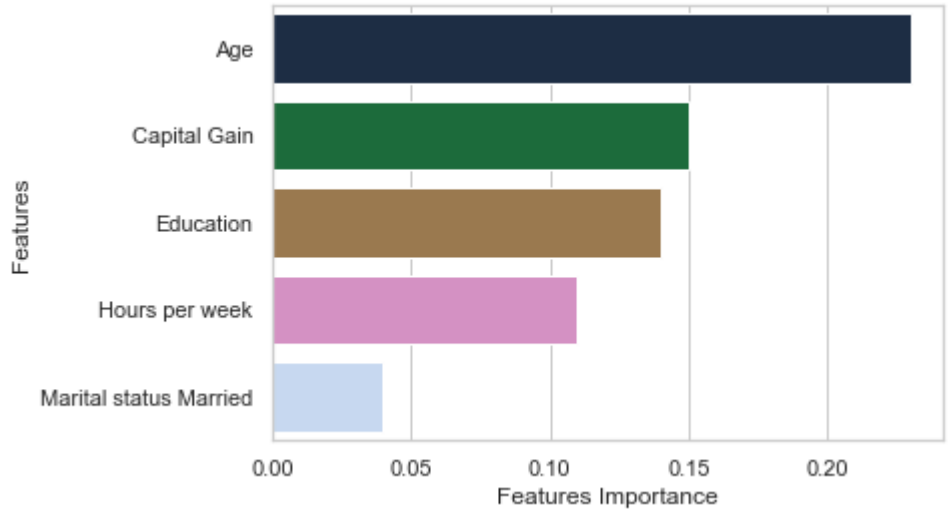
After doing all of the model fitting, we applied to the test set our Gradient Boosting Classifier, and we had a prediction of 19.79% (2000) people with income above average, and 80.21% (10100) people below.

5. DISCUSSION

As we could see from the results, it seems that the features that influence most the income level in our final model are the married marital status, the years of education, the age and the working hours per week. We can also highlight the importance of the feature “money received”, which was related to the people that were considered essential by the State and were paid to go to Newland. The ticket price feature which is related to the people that paid to enter Newland is also important, even if it has half of the importance of Money received. This may imply that the people that were paid by the Government to enter Newland have double the influence on the prediction than the people that entered the new Planet by paying.

We found some similar conclusions regarding feature importance in another study named *“Using decision tree classifier to predict income levels”* in which some scientists were predicting income levels of individuals based on some attributes.[13] The most important ones in their study are shown in Figure 5-1:

Figure 5-1: Feature Importance of Random Forest (BEKENA, 2017)



We can see that the most important features are similar to our model, represented in Figures 4-4 and 4-11. By comparing these two studies, we can conclude that Age is really important for the level of income, as well as years of education: it seems that in both models the more a person is educated the higher will be the income. Working hours per week is also important: it seems the more a person works, the higher the income.

6. CONCLUSION

As stated in the introduction the objective of this project was to make a predictive model for the income of new residents, so that the Newland government had an idea of the amount of taxes it was going to receive. This was successfully done by using gradient boosting classifier to predict the income class of each individual.

The classifier was used due to the problem nature and also because among all the other models it was the one with the highest F1 score('micro') 87.4% and also with the highest accuracy 84.4%. As the majority of individuals receive a lower income than the average the model had more difficulties in classifying the ones from the higher income class with an error of 33.3%. Despite this fact, the model was really useful to understand which features were more important to describe the individuals with lower or higher incomes than average. The results from our model show that marital status, the amount of money received to go on the mission, the education level, the ticket price and age were the best to explain the differences between the two classes. From the test set in which we had to predict the different income classes, we predicted that 20% of the individuals had a higher salary than average and 80% lower.

7. REFERENCES

1. Manohar Swamynathan. Mastering Machine Learning with Python in Six Steps - A Practical Implementation Guide to Predictive Data Analytics Using Python – Second Edition, 2019.
2. https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/ (SMOTE)
3. https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html
4. https://www.youtube.com/watch?v=jXSw6em5whI&t=303s (RFE)
5. https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/?fbclid=IwAR3jJmuve5_ymQsh5urS7TKFw8ZP35Kr3hvXhqLNSJpGGnvdTrmITXtk8-I (GradientBoosting)
6. https://towardsdatascience.com/preprocessing-encode-and-knn-impute-all-categorical-features-fast-b05f50b4dfaa
7. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
8. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
9. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
10. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
11. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html
12. https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html
13. https://mpra.ub.uni-muenchen.de/83406/1/MPRA_paper_83406.pdf
14. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html