Newland: Machine Learning Project

Carolina Rocha (R20170740), Filipe Lourenço (R20170799), Guilherme Neves (R20170749), Mariana Martins (R20170804), Mariana Gonçalves (R20170780)

Abstract: the purpose of this project is to successfully predict if a citizen has an income above average or not, by building models able to do it. To do so, the available data were prepared and cleaned, and the most relevant features were selected. Several machine learning algorithms were tested to reach the best combination of parameters and higher accuracy scores. In the end, we concluded that the ensemble Gradient Boosting 1 classifier was the one that present the best results. Thus, we chose this one to predict the classes of the dependent variable.

Key words: Data; Variables; Predictive models; Score; Algorithm; Training; Validation; Ensemble Classifiers; Overfitting

I. Introduction

A mission, called "Newland", was launched in 2046, aiming to inhabiting a new planet that was discovered with similar conditions to Planet Earth. Therefore, there were 3 populated spaceships, each one with a capacity of about 40,000 people to that new planet.

In order to make the city built in the new planet more financially sustainable, the Newland government decided that the new inhabitants should start paying taxes. Therefore, based on a binary feature called 'Income', a person should pay 15% of his/her income if he/she has an income below or equal to the average, or 30% of his/ her income otherwise. In order to predict to which class each person belongs to, the Newland government invited our group to build a predictive model to do so.

We were provided a database of 32500 people so then we could split this data in two datasets: one with 22400 observations, which is going to be used to build our machine learning models and to assess the performance of them by creating a validation dataset. The other one composed by 10100 observations, is going to be used to assess if the chosen model performs well or not on unseen data, by fitting the chosen model to this data to then predict to which class of the dependent variable 'Income' each observation belongs to.

II. Background

<u>One Hot Encoding</u>

The OneHotEncoder from scikit-learn transforms the categorical variables into binary ones, by taking as input "an array-like of integers or strings, denoting the values taken on by categorical (discrete) features" (scikit-learn.org, 2020). In other words, the categories for each categorical feature must be assigned to integer values, representing each one of them a binary vector, filled in with ‘0’ except the positions where the integer value occurs, which correspond to ‘1’.

The OneHotEncoder has some important hyperparameters to set, such as:

- **‘Drop’:** controls which categories or binary features to be created by OneHotEncoder. For instance, it allows to retain all features (‘Drop’ equals ‘None’); To not include the first category of a feature, which causes to not create it, avoiding multicollinearity (‘Drop’ equals ‘first’); specify which category of each feature to be dropped (‘Drop’ equals ‘drop[i]’); eliminate the first category in every feature which has two categories (‘Drop’ equals ‘if_binary’).
- **‘Sparse’:** If set to ‘True’, OneHotEncoder returns a sparse matrix with all one-hot encoded categorical features, being each column assigned to a different one. If set to ‘False’, it generates an array, where each row represents a different binary feature.

This type of encoding is very useful for transforming categorical data due to many machine learning algorithms not accepting categorical data as an input and to improve the interpretability of the categorical features.

MIC (Mutual Information Coefficient)

Mutual Information is often used to measure the mutual dependence between two variables. More specifically, it quantifies the "amount of information" about one random variable through observing the other random variable. This value is always larger than or equal to zero. The larger the value the stronger the relationship between the two variables. If the value is zero, it means that the variables are independent.

Passive Aggressive Model

Passive-Aggressive algorithm is somewhat similar to a Perceptron model, in the sense that it does not require a learning rate. However, contrary to the Perceptron, it includes a regularization parameter C (maximum step size), that denotes the penalization the model will make on an incorrect prediction. A higher C value yields stronger aggressiveness, while lower values allow a better adaptation to it.

The algorithm is passive if the prediction is correct because it keeps the model and does not make any changes. In case the prediction is incorrect, changes are made to the model, so it is considered Aggressive. The core concept is that the classifier adjusts its weight vector for each mis-classified training sample it receives, trying to get it correct. So, the weighted vector that is multiplied by the input vector will slowly forget the previous classifier and learn the new distribution.

Support Vector Machine model is also used here since the Hinge loss function is the base of the algorithm for discovering the "maximum-margin" classification. A good robustness of the model is necessary, otherwise, too rapid changes produce consequent higher misclassification rates.

Nearest Centroid Classifier

The Nearest Centroid Classifier works on a simple principle: given a data point (observation), the Nearest Centroid Classifier simply assigns it the label (class) of the training sample whose mean or centroid is closest to it.

The way the K nearest centroid works is quite easy, first, while training, the centroid for each target class is computed. Then, after training, the distances between the point and each class' centroid is calculated and the minimum distance for each point is picked, so the centroid to which the given point's distance is minimum, its class is assigned to the given point.

Voting Classifier

Voting Classifier is a machine learning model that aggregates several classifiers to make a prediction, based on the highest majority of voting. It is not an actual classifier but a wrapper for a set of different ones to classify the data with combined voting. The ensemble, by combining different algorithms, is ideally stronger than any of the individual models alone. It is important to include diverse classifiers, to be sure that the error made by one might be resolved by another one and that they do not follow similar mathematical techniques.

Voting Classifier has two hyperparameters: estimators, that create a list for the classifiers while assigning names, and voting, that can be hard or soft. In hard voting the predicted output class is the class with the highest majority of votes. In soft voting, the output class is the prediction based on the average of probability given to that class. In practice the output accuracy will be higher for soft voting as it is the average probability of all estimators combined. When the number of classifiers is even, it is recommended to use soft voting, when it is odd hard voting.

It is very useful when we do not know which classification method we should use, when a single method shows bias towards a particular factor, or whenever a single strategy is not able to reach the desired accuracy threshold. One advantage of Voting Classifier is that by aggregating several models, is possible to overcome the disadvantages of each model to generalize the classification model.

III. Methodology

Firstly, the training dataset was imported, and we proceeded to analyze the type of variables and some basic statistics of each one of them. We also checked the proportion of each class of the dependent variable 'Income' to have a deeper knowledge about the variable we will predict.

Before proceeding to the transformation and creation of new variables, we set the index of the dataset to be 'Citizen_ID' and replaced '?' with NAN in order to Python identify these values as missing values.

Following that, we proceeded to the transformation and creation of the following variables:

- **'Age'**: numerical variable created from the variable 'Year of Birth', so it would be easier to perform meaningful analysis based on the age of each person.
- **'Group B'**: binary variable created from the variable 'Money Received', holding the value '1' if a person received money to take part in the mission and '0' otherwise.
- **'Group A'**: binary variable created from the variable 'Money Received' and 'Ticket Price', holding '1' if a person didn't receive any money and didn't pay anything to participate in the mission (volunteers).
- **'Male'**: binary variable based on the feature 'Name', which holds the title for each person (Mr., Mrs. and Miss). Male equals '1' if 'Name' equals 'Mr.' and '0' otherwise.
- **'Higher Education'**: binary variable built from the variable 'Years of Education', which has the values '1' or '0', whether a person has more than 13 years of education or not.

Note: We did not create variables 'Group C', 'Female' and 'Not Higher Education' to avoid having perfect multicollinearity.

Finally, we dropped out the variables 'Birthday', 'Year of Birth', 'Name' and 'Education Level' from our dataset, since they became redundant after creating the new features mentioned above.

Incoherencies

Following the transformation and creation of new variables we decided to search for incoherent data, in order to not negatively influence the results of our machine learning models. Therefore, once those incoherent data were found, the corresponding records were deleted from the dataset.

The incoherencies tested were the following:

- **Incoherence 1**: If a citizen has a 'Working Hours per week' column value higher than 0, then 'Employment Sector' must not be 'Unemployed' or 'Never Worked'.
- **Incoherence 2**: If the 'Marital Status' is 'Married - Spouse Missing' or 'Married - Spouse in the Army' or 'Single' or 'Divorced' or 'Separated' or 'Widow', then the 'Lives with' column value must not be 'Wife' or 'Husband'.
- **Incoherence 3**: If the role of a person is 'Army', then 'Employment Sector' must be 'Public Sector - Government'.
- **Incoherence 4**: In our understanding, we considered that the feature 'Age' could not hold values higher than 120 years old and according to the given instructions it cannot accept values lower than 17 years old.
- **Incoherence 5**: A person cannot have more 'Years of Education' than years of life ('Age').
- **Incoherence 6**: A citizen cannot be considered to be a volunteer ('Group A' equals '1') and, at the same time, to have received money to take part in the mission ('Group B' equals '1').

Data Partition

In this step, we splitted the data into two datasets: the training dataset and the validation dataset, by randomly selecting different observations to each dataset, while ensuring each one has the same proportion of each class of the dependent variable.

The purpose of the training dataset is to fit a predictive model to it, which has the ability to generalizes well to new and unknown data. At the same time, the goal of the validation dataset is to fine tune the hyperparameters of the model, which was fitted to the training data, in order to guarantee that this supervised model correctly predicts the labels of the dependent variable on unseen data. It also allows to check for existing problems, such as overfitting.

Keeping the assumptions above in mind, we decided to follow the standard approach of splitting the original dataset into a training dataset comprised of 70% of the original observations and a validation dataset containing the remaining observations. Moreover, we also defined the variable 'Income' as the target variable, while separating it from the original dataset.

Note: all the above steps were implemented to the Test dataset.

Outliers

Following the data partition, our group focused on searching for extreme observations, data with either low values or high values when compared to the remaining data. From the describe function done previously, we noticed that there were two variables that presented extreme values. Therefore, two box plots and two histograms showing the distribution of the data for the variables 'Working Hours per week' and 'Years of Education' were plotted in order to search for univariate outliers.

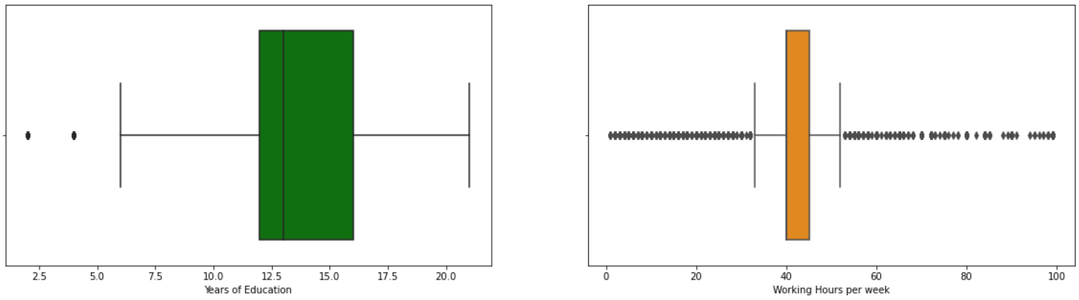


Figure 1- Boxplots of the variables 'Years of Education' and 'Working Hours per Week'

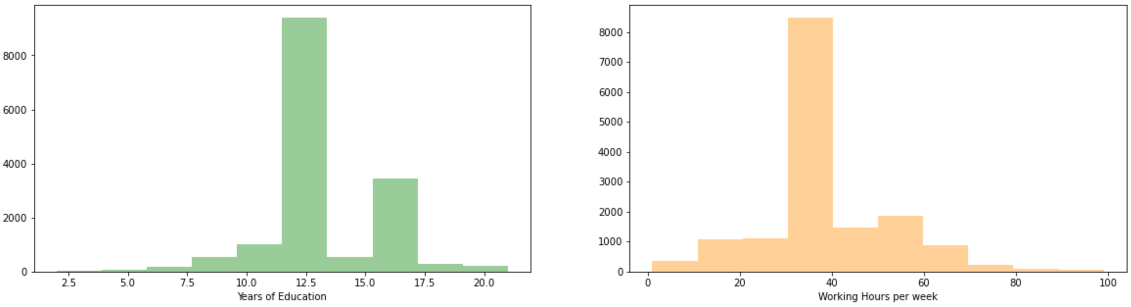


Figure 2- Histograms of the variables 'Years of Education' and 'Working Hours per Week'

After this, we tried to remove univariate outliers for these two features using the Inter-Quartile Range (IQR) method, but we verified that too many observations were removed (above 3% of the data were dropped out).

Thus, we chose to remove univariate outliers manually, by removing all observations which were not in the interval between 5 and 21 for the variable ‘Years of Education’, and in the interval between 10 and 80 for the variable ‘Working Hours per week’. In the end, we dropped out 2.7% of the total observations from the training dataset. Furthermore, we built graphs showing the Pairwise Relationship of the numerical variables so we could find any bivariate outliers, but we opted not to eliminate any of those as we did not find any extreme observation.

Missing Values

In this part, we searched for features in both training dataset and validation dataset which had missing values. Accordingly, we verified that there were 3 variables with missing values: ‘Base Area’, ‘Employment Sector’ and ‘Role’. For the first 2 variables, our decision was to fill in the missing values with the mode, since there was a category for each one which had a very high frequency compared to the other ones.

On the other hand, due to all categories in ‘Role’ having similar frequencies, a different approach was used to fill in the missing values of ‘Role’, by applying the KNN Imputer. This algorithm finds the nearest neighbor for each observation with a missing value in the variable ‘Role’ (in this case), based on the values for the other numeric features. Then, this algorithm replaces the missing value in ‘Role’ with the value of the nearest neighbor for that same feature.

One Hot Encoding

The dataset had a lot of categorical variables and we wanted to include this type of variables in the predictive models, since they can provide important information to them and improve their prediction ability. Thus, we transformed these features into binary ones using the One-Hot Encoding algorithm. Before proceeding to this step, it was needed to aggregate each label of the categorical variables into several main ones, in order to reduce the cardinality of each variable, in both training and validation datasets. Moreover, not all of the newly created binary variables were added, due to the existence of perfect multicollinearity between all the binary variables built from the same variable. Therefore, one of the binary variables created from each one of the original categorical variables was dropped.

In the table below, it can be checked which categories were aggregated and which new categories were built for each categorical feature:

Feature	Old Categories	New Categories
Base Area	‘Northbury’ & More 39 categories	‘Northbury’ & ‘Not Northbury’
Marital Status	'Single','Divorced','Widow', 'Married - Spouse Missing', 'Separated', 'Married - Spouse in the Army' & 'Married'	‘Alone’ & 'Together’
Lives with	'Children', 'Other Family', 'Wife', 'Husband', 'Other relatives' & 'Alone'	‘Lives with someone’ & ‘Lives Alone’
Employment Sector	'Self-Employed (Company)’, 'Self-Employed (Individual)' & ‘Public Sector - Others', 'Public Sector - Government' & 'Private Sector – Services’, 'Private Sector - Others',	‘Self-Employed’ & ‘Public Sector’ & ‘Private’
Role	'Repair & constructions', 'Agriculture and Fishing', 'Cleaners & Handlers', 'Household Services' & 'Security', 'Sales', 'Management', 'Transports', 'Machine Operators & Inspectors', 'Army' & 'Professor', 'IT', 'Administratives', 'Management' & 'Other services”’	‘Low qualification Jobs’ & ‘Medium qualification Jobs’ & ‘High qualification Jobs’ & ‘Other services’

Figure 3- Table with old and new categories

Data Standardization

Data standardization is a fundamental step in data preprocessing, given that the various variables have different scales, and these differences may cause problems in the models that are based on the distances between the observations, such as K-Nearest Neighbors or Support Vector Machine. So, right after applying One Hot Encoding, we standardized only the non-binary data since it does not make sense to apply it to variables whose values simply mean the presence or absence of that variable.

Therefore, we decided to use four different techniques for scaling our data for both training and validation datasets. The first two were the MinMax Scaler between 0 and 1 and the MinMax Scaler between -1 and 1 that transform the values by scaling them between the given range. The third was the Standard Scaler that transforms the values in order to each feature to have 0 mean and 1 standard deviation and the fourth was the Robust Scaler which scales the values using statistics that are robust to outliers, such as the median and the quantiles.

To choose the best scaling method to use, we decided to do the modelling first, applying the four techniques for the diverse classifiers, and in the end choose the technique with better results that was the Standard Scaler for most of the models.

Feature Selection

Feature selection is a fundamental step before start modeling, since it decreases training time, reduces the risk of overfitting, minimizes the curse of dimensionality and helps increasing interpretability. Taking in consideration several methods in this step, we decided to use 8 methods, considering the results of all of them when deciding the variables to keep.

The methods used were the following:

- **Pearson Correlation,** which measures the linear relationship between the variables. In the image below we can see that Years of Education is highly correlated with Higher Education as expected, Group B is positively correlated with Money Received and is extremely negatively correlated with Group A, since they are binary features, and Group A is negatively correlated with Money Received and Ticket Price.

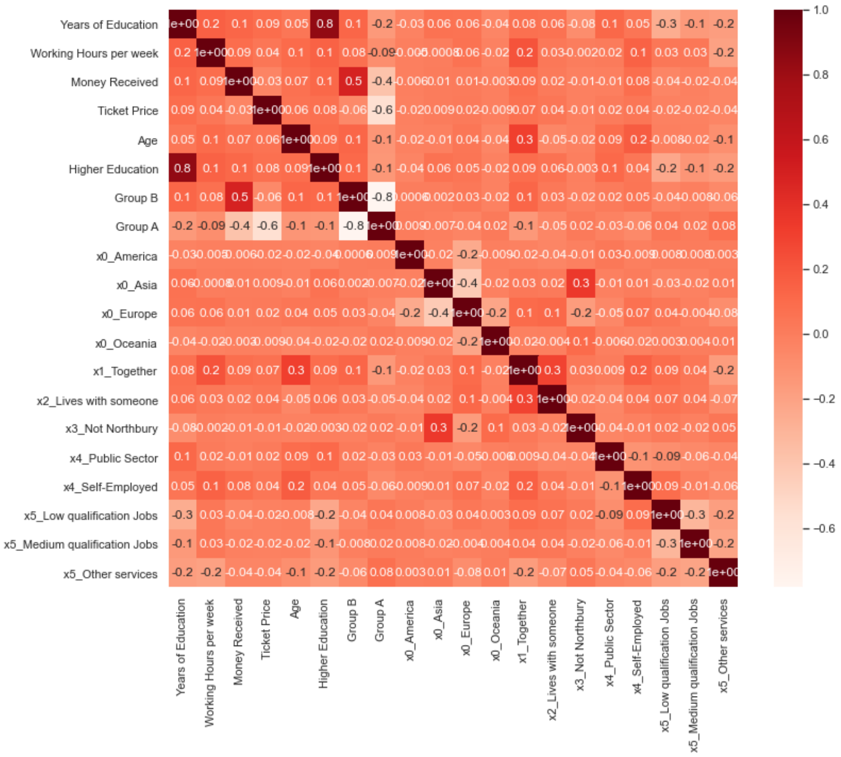


Figure 4- Pearson Correlation Plot

- **Spearman Correlation,** which measures the rank correlation, assessing if a relationship between two variables is similar to a monotonic function or not. Here, the variables that are correlated between them are similar with what occurred in Pearson, the major difference is that some values are closer to 1.

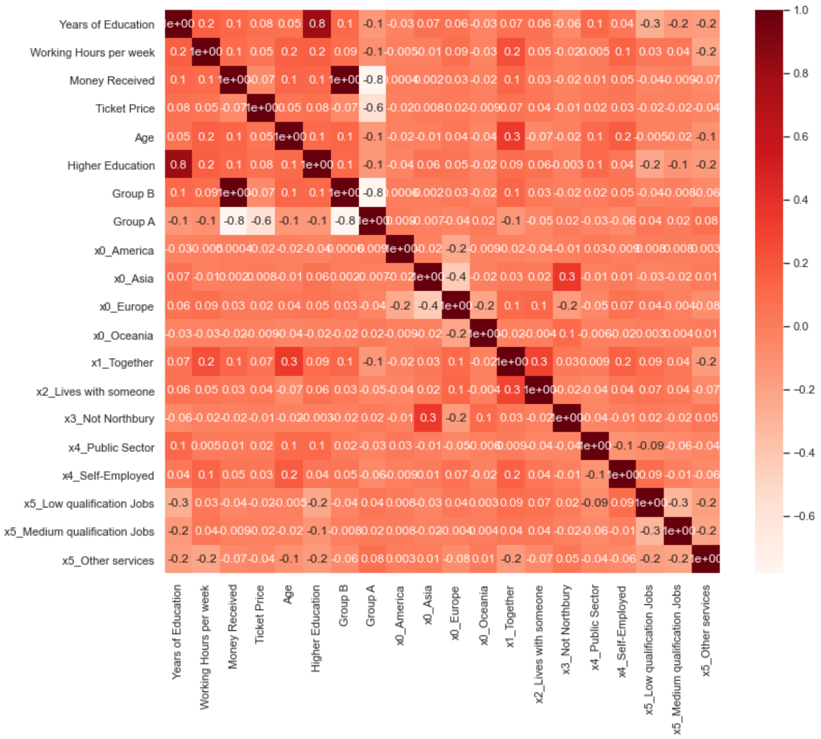


Figure 5- Spearman Correlation Plot

- **Lasso Regression,** that puts a constraint on the sum of the absolute values of the model parameters. The sum must be less than a fixed value. In order to do so, the method applies a shrinking (regularization) process where it penalizes the coefficients of the regression variables shrinking some of them to zero. A zero coefficient in Lasso Regression normally implies that the respective variable has very few or no importance on the model.

- **Ridge Regression**, also applies a shrinking process that penalizes the coefficients. The main difference is how this regularization is applied, since in Ridge the coefficients become very close to zero, but never zero. Similar to Lasso, variables with the highest coefficients are considered the most important ones, and vice versa.

- **Recursive Feature Elimination,** at the beginning, the estimator is trained on the entire set of features and the importance of each feature is obtained either through the respective coefficient or through a ranking of the variable in comparison with the remaining ones. Then, the least important features are pruned from the current set of features. This procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

- **MIC (Mutual Information Coefficient),** measures the mutual dependence between two random variables, that is, the level of correlation between them. The values are higher or equal to zero, where larger values indicate a greater dependence between the two variables.

- **Weights of dependent variable on binary features,** where we visualize the absolute number and the proportion of zeros and ones relatively to the target variable for each binary variable, in order to evaluate the discriminant capacity of each binary variable. Variables with bigger differences between zeros and ones would be more important for classification.

- **Decision Tree,** measures the importance of each variable giving a score based on the reduction in Gini Index or Entropy to select split points. In this method, we compared both Gini Index and Entropy, showing similar values for each variable where higher values indicate a greater importance.

The results of this methods are presented in the following table:

Variables/Feature Selection Tools	Lasso Regression	Ridge Regression	RFE	MIC	Weights	Decision Tree	Final Decision
Years of Education	🟩	🟩	🟩	🟩		🟩	KEEP
Working Hours per week	🟩	🟩	🟩	🟩		🟩	KEEP
Money Received	🟩	🟩	🟩	🟩		🟩	KEEP
Ticket Price	🟩	🟩	🟩	🟩		🟩	KEEP
Age			🟩	🟩		🟩	KEEP
Higher Education	🟥	🟥	🟥	🟩	🟩	🟥	DROP
Male	🟥	🟥	🟥	🟩		🟥	DROP
Group B	🟩	🟩	🟩	🟩	🟩	🟥	KEEP
Group A		🟩	🟥	🟩	🟩	🟥	DROP
x0_America	🟥	🟥	🟩	🟩		🟥	KEEP
x0_Asia	🟥	🟥	🟥	🟩	🟥	🟥	DROP
x0_Europe	🟥	🟥	🟥	🟩	🟥	🟥	DROP
x0_Oceania	🟥	🟥	🟩	🟩		🟥	KEEP
x1_Together	🟩	🟩	🟩	🟩	🟩	🟩	KEEP
x2_Lives with someone	🟥	🟥	🟥	🟩		🟥	DROP
x3_Not Northbury	🟥	🟥	🟥	🟩	🟥	🟥	DROP
x4_Public Sector	🟥	🟥	🟥	🟩	🟥	🟥	DROP
x4_Self-Employed	🟥	🟥	🟥	🟩	🟥	🟥	DROP
x5_Low qualification Jobs			🟩	🟩	🟥	🟥	KEEP
x5_Medium qualification Jobs	🟥	🟥	🟥	🟩	🟥	🟥	DROP
x5_Other services	🟥	🟥	🟩	🟩	🟩	🟥	KEEP

Figure 6 - Table with feature selection conclusions

Modelling

After making feature selection, we dropped some variables, in order to just use the variables we believed to be the most important to the prediction and models.

Then, we tested several different algorithms: Linear Regression, Logistic Regression, Neural Networks, Decision Trees, K Nearest Neighbors (Tree and Ball Tree), K Nearest Centroids, Passive Aggressive, Naive Bayes and Support Vector Machine.

The main goal was to optimize the score and average score. To do that, we defined a function to calculate the average score and standard deviation of the training and validation set, to check if the model successfully predicts the dependent variable and if they vary a lot or not.

Several standardization techniques and cross-validation techniques were tested to reach the best results. We decided to use Repeated K Fold to split the data and the standardization method Standard Scaler.

Ensemble

After testing all the models and finding each one performed better, we started the ensembles. Ensembles uses multiple learning algorithms to improve predictive performance so to try to achieve that we decided to build different classifiers. The models tested in the classifiers where the ones that presented the best scores in the previous step.

Bagging Classifiers, Ada Boost, Random Forest, Voting Classifier, Gradient Boost and Stacking Classifiers were the ensembles applied in the phase.

We decided the best classifier by the scores of the train and validation and when possible, with the average score, function previously created. Here, we took into a lot of consideration the average score because it tells us the standard deviation, so we were able to understand the behavior of the ensemble.

IV. Results / Discussion

The models that presented the best results were the Neural Networks, Decision Tree, K Nearest Neighbor Tree, K Nearest Neighbor Ball Tree and Support Vector Machine.

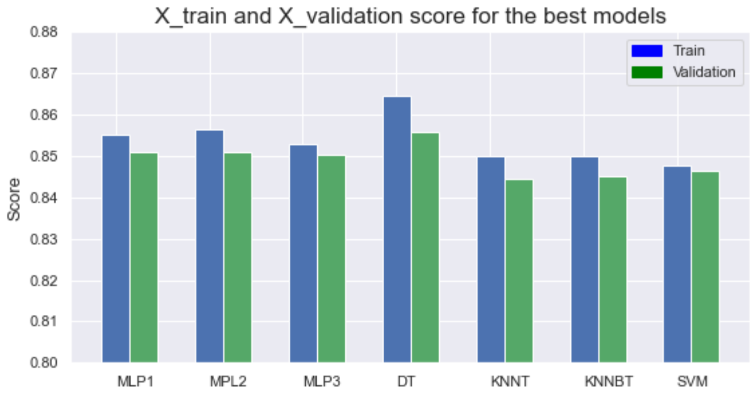


Figure 7- X-train and X-validation Score for the Best Models

In these models we had to take into consideration the large range of the parameters to use and, simultaneously, the computational efficiency.

When modelling the Decision Tree, we had in consideration that decision trees tend to overfit in data with a lot of features. So, our goal was not only to find the optimal value of effective alpha to reach the best score in train and validation, but also make sure that the scores on train and validation were similar. When we set the criterion to entropy the model presented overfitting, so we left it as default (gini). The most important parameter was the ccp_alpha because as alpha increases the more of the tree is pruned creating a decision tree that generalizes better.

The most important hyperparameters in K Nearest Neighbor Tree and K Nearest Neighbor Ball Tree are 'n_neighbors' and 'weights'. Thus, those two parameters were the ones to be fine-tuned until a desirable average score for both train and validation datasets was achieved. The first one relates to the number of neighbors to consider in order to classify new observations. Accordingly, it can lead to overfit, if given a too low number of observations, or to underfit, if given a too large number of neighbors, thus 50 was the number found out to be the one which would give neither of them. On the other hand, the parameter 'weights' refers to whether all points in each neighborhood are weighted equally ('weights' = 'uniform') or if closer neighbors have a greater weight than the ones further away ('weights' = 'distance'). In this case, the first option was chosen, as it leads to better average scores in both train and validation datasets.

In Support Vector Machine one of the most important parameters to tune is the C. C works as a regularization parameter and tells the SVM optimization how much we want to avoid misclassifying each training example, the larger the value the smaller the margin hyperplane. Since we concluded that the best kernel for us was the poly, we defined the degrees and the gammas, specific for this kernel.

In Neural Networks models, more complex models with a higher number of hidden layers, a higher batch size or a higher number of maximum iterations did not get better results.

Besides that, increasing the complexity of the model by increasing these parameters, would be computational heavier and, consequently, would increase the time to run those models. After trying several parameters, we got some overfitting, so, to solve this problem we had to change some parameters such as the alpha, the regularization term, or the momentum for gradient boosting update in Neural Networks.

Mostly, in Ensemble models we got a higher score when comparing with the previous models such as K-Nearest Neighbor Tree, Decision Trees or Neural networks, so these represent most of the models submitted in Kaggle.

The ensembles that presented the best results were the Bagging Classifiers, Ada Boost, Voting Classifier, Gradient Boost and Stacking Classifier.

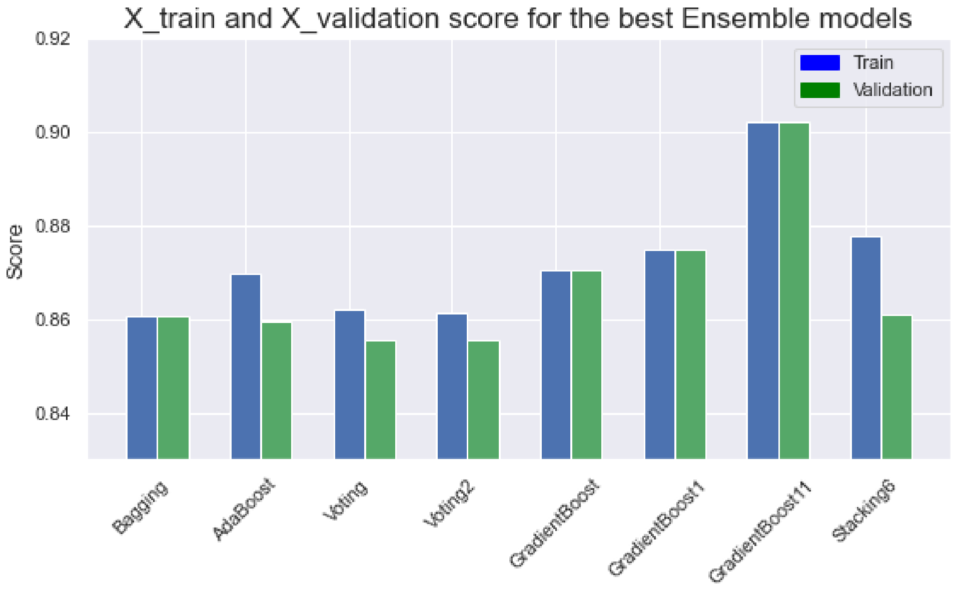


Figure 8- X-train and X-validation Scores for the Best Ensemble Models

Regarding Bagging Classifier, we applied cross-validation and checked the mean score for different base estimators (the models that presented the best results), where the highest score corresponded to a Decision Tree. In the end, we submitted two Bagging Classifiers in Kaggle, one with a Decision Tree as base estimator and the other with a Neural Network, and the Decision Tree continued to present the highest score.

Concerning AdaBoost Classifiers we applied a Grid Search and then some hyperparameters were manually fine-tuned. The ones thought to be the most important to achieve a better average score for both train and validation datasets were 'n_estimators' and 'learning_rate'. The first one is related with the number of models to be added to predict the training dataset therefore, the more classifiers are added, the more accurate the predictions are. Despite that, as the number of 'n_estimators' was being increased, results did not improve much and the computational cost of the algorithm was getting more and more expensive. Therefore, we set 35 as the number of estimators which could lead to good predictive ability, as well as a low-medium computational cost. The ‘learning_rate’ was firstly set to be between 0.1 and 1 in the Grid Search but we verified after several tests that the best learning rate was 0.021, being this number the final ‘learning_rate’. After testing several combinations of the hyperparameters belonging to AdaBoost, we submitted two AdaBoost Classifiers in Kaggle with different base estimators, one with Decision Tree and other with a Random Forest and the first one registered a higher score.

Relatively to Voting Classifiers, we also applied a GridSearch and then, we tried different combinations between the estimators, applying cross-validation to each one and choosing the two classifiers that got the highest mean score, submitting these two in Kaggle. The ‘estimators’ hyperparameter accepts models that were tested previously, in our case K Nearest Neighbor Ball Tree, Decision tree and Neural Network 3 were the ones used for the first submission in Kaggle, and a second submission with the same last two models but with an ensemble - Random forest. Another hyperparameter, that was taken into consideration was the 'voting', where the predicted class labels to be used can be 'hard' or 'soft', in our case the 'soft' method was the one applied.

About Gradient Boosting Classifiers, we applied a GridSearch and cross-validation to choose the best classifiers. With these classifiers we got a score slightly better than the remaining, being verified in Kaggle, where the three classifiers with the highest score are Gradient Boosting Classifiers. We also noticed that increasing the number of estimators or the learning rate would cause overfitting in the models, therefore it was fundamental to find a balance between these two parameters and the accuracy of the classifier.

In Stacking Classifiers, we could not apply a GridSearch, since it would take a long time to run, so we had to understand what combinations of classifiers would get better results, the base estimators, and which classifier would be better to combine the base estimators. In the end, we got three different classifiers, where it was not possible to apply cross-validation and submitted them in Kaggle. The difference between these three was the base estimators and final estimator. Between them the estimators change from a combination of all the best models and a combination of the previous built ensembles. The other difference, when the ensembles have the same estimators base, is the final estimator changing from Logistic Regression to Neural Networks.

After evaluating the average scores for training and validation datasets for all models and submitting the predicted values for the dependent variable of the test dataset on Kaggle, we chose ‘Grad_Boost1’ to be the model through which unseen data would be predicted. Therefore, we decided to apply some performance measures on it, such as precision, recall, f1-score, support and confusion matrix. By analyzing this metrics, we can conclude that our final model is very good at predicting the ‘0’s of the dependent variable, although it is not so good at predicting the ‘1’s.

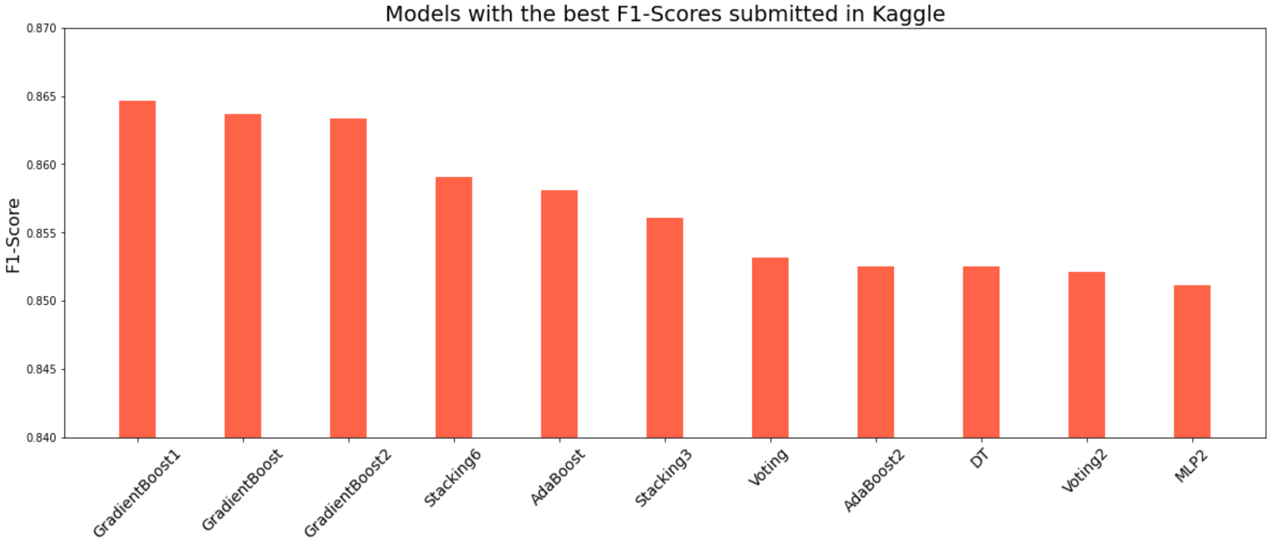
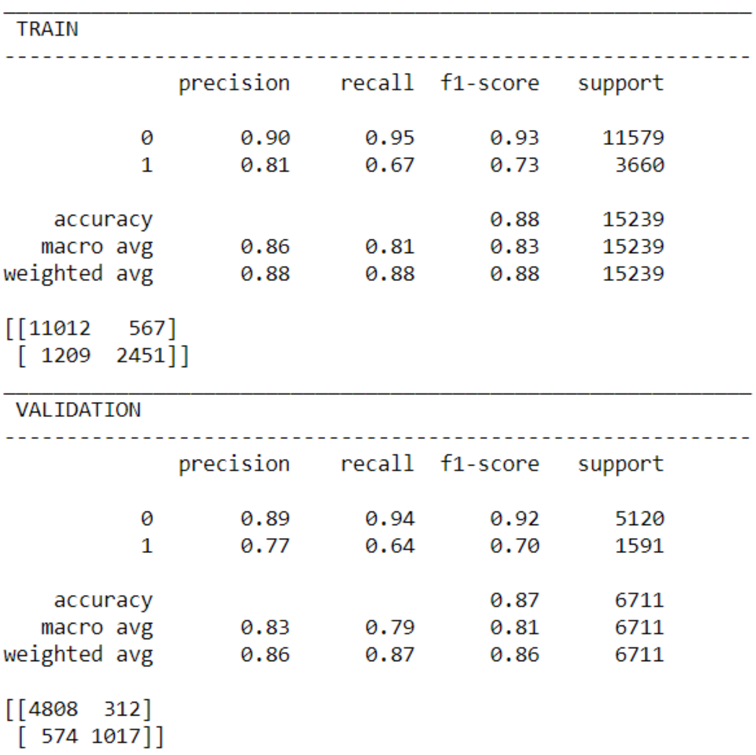


Figure 9- Models with the best F1-Scores submitted in Kaggle



```
________________________________________________________________
 TRAIN
----------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.90      0.95      0.93     11579
           1       0.81      0.67      0.73      3660

    accuracy                           0.88     15239
   macro avg       0.86      0.81      0.83     15239
weighted avg       0.88      0.88      0.88     15239

[[11012   567]
 [ 1209  2451]]
________________________________________________________________
 VALIDATION
----------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.89      0.94      0.92      5120
           1       0.77      0.64      0.70      1591

    accuracy                           0.87      6711
   macro avg       0.83      0.79      0.81      6711
weighted avg       0.86      0.87      0.86      6711

[[4808  312]
 [ 574 1017]]
```

Figure 10- Performance Measures of Gradient Boost 1

V. Conclusion

During the project, our group faced some challenges related with the dataset used and the complexity of all the process, since the dataset importation until the choice of the best model is a process where we need to move forward and backwards, in order to ensure that we use the techniques and models that best fit to our data.

After data importation and data exploration, the following step was to prepare and transform the data, as it had a crucial role in our project since a rigorous analysis and the transformations made here would have huge impact in the further steps.

Before start modelling, we selected the most important variables taking in consideration various techniques and combining their outcomes. With the variables selected we started modelling where we used different classifiers such as Neural Networks, Decision Trees, k-Nearest Neighbor, Passive Aggressive, among others. In the end, three different Neural Networks and one Decision Tree got the highest scores that we tested later on Kaggle. We also used some ensemble classifiers such as Bagging, Random Forest, AdaBoost or Gradient Boosting where Gradient Boosting classifiers had the highest score.

We submitted the classifiers that got the best scores in Kaggle and, from these, the ones that had the highest scores in Kaggle, we decided to run them and submit them in Kaggle several times to ensure they were consistent. Finally, we chose Gradient Boosting 1 classifier, which had the highest score between all the submitted models and a low variance.

Therefore, our group was able to build a final predictive model to be presented to the Newland Government, in order to assess if a person has an income below or above the average, to then apply a different tax rate based on that.

VI. References

Brownlee J., 2017, *How to One Hot Encode Sequence Data in Python,* 2020,

< https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/>

2017, *ML Algorithms addendum: Passive Aggressive Algorithms*, 2020,

<ML Algorithms addendum: Passive Aggressive Algorithms - Giuseppe Bonaccorso>

2020, *Passive Aggressive Classifiers*, 2020,

<Aggressive Classifiers - GeeksforGeeks>

sklearn.neighbors.NearestCentroid , 2020,

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestCentroid.html>

1.6. Nearest Neighbors, 2020,

<https://scikit-learn.org/stable/modules/neighbors.html>

2019, *ML | Voting Classifier using Sklearn*, 2020,

<ML | Voting Classifier using Sklearn - GeeksforGeeks>

Brownlee J., 2020, *Stacking Ensemble Machine Learning With Python*, 2020, <https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/>

sklearn.ensemble.StackingClassifier, 2020,

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html>