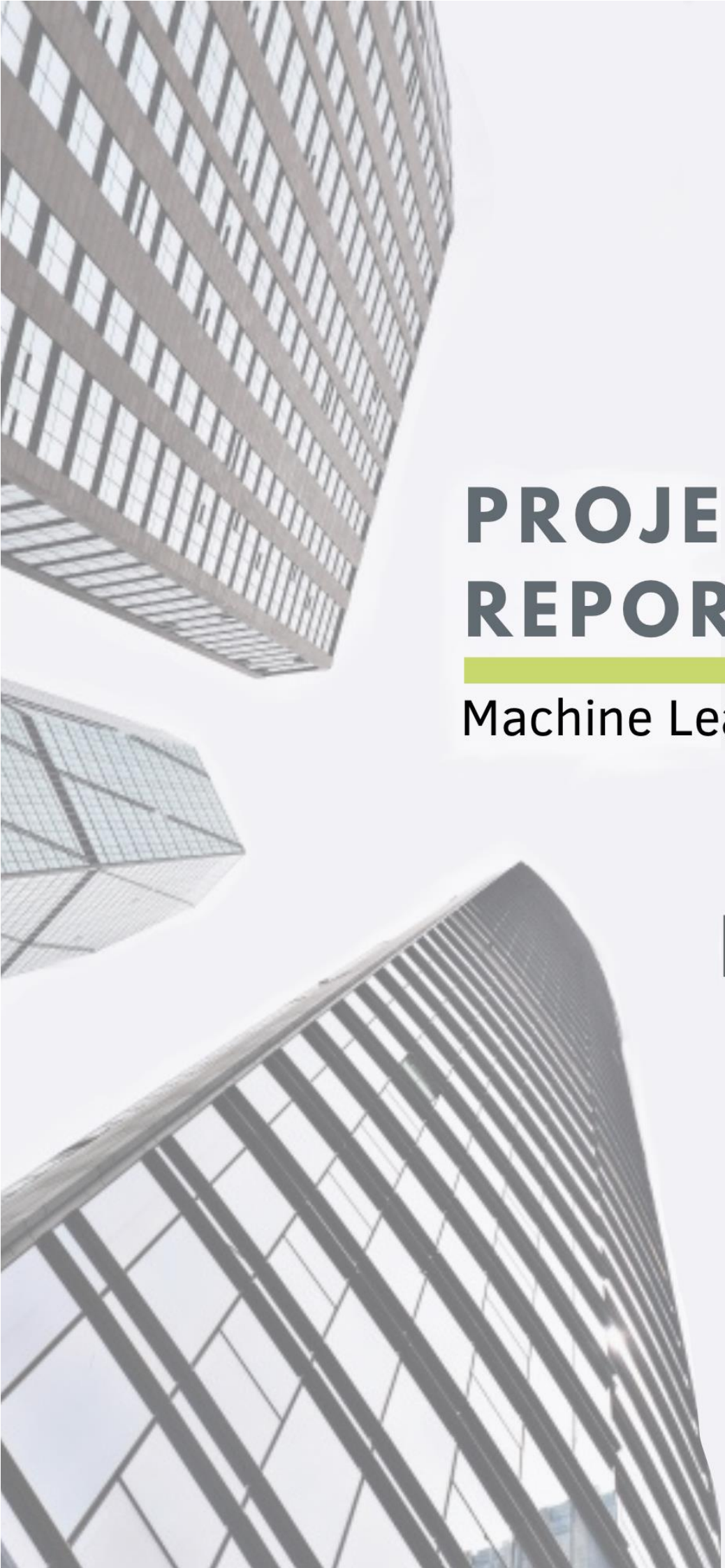


PROJECT REPORT

2020/21

Machine Learning

NEWLAND

Ana Carrelha
m20200631@novaims.unl.pt
Inês Melo
m20200624@novaims.unl.pt
Inês Roque
m20200644@novaims.unl.pt
Tomás Gonçalves
m20200584@novaims.unl.pt

Abstract— Following a supervised machine learning methodology, for this project, we were proposed to predict whether the income of a percentage of people is below or above the average, based on a train data set in order to achieve the best score possible. By preparing data, separating target from the rest of the data, filling missing values, dealing with outliers, preparing features, constructing the most common predictive models, training data, discuss predictions scores from more than one view and improving them by doing hyperparameters tuning and resort to search models, we are able to find the best model with the best parameters and the highest score with low overfitting to reach our purpose and give us a higher confidence for labelling unlabelled data. We ended up with a good score on Kaggle, using the predictive model XGBoost.

Keywords: Project, Machine Learning, Predictive Models, Income, Hyperparameters tuning, Supervised, Features, Score.

I. INTRODUCTION

This project was developed as a part of Machine Learning course, for the Master's in Data Science and Advanced Analytics. Based on a 2048 dataset with 22400 observations, we were proposed to create a predictive model for a futuristic city called Newland.

Our purpose is to predict if the income of the population, that is on their way to Newland, is above or below the average, so that the government is able to know the tax rate to apply to each citizen.

After the predictions, the model will be applied to 10100 future citizens, that are represented in the test dataset.

Along this report, we will study several models to find which one fits better to our problem.

II. BACKGROUND

- *"**Random search** is a technique where random combinations of the hyperparameters are used to find the best solution for the built model. It is similar to grid search, and yet it has proven to yield better results comparatively. The drawback of random search is that it yields high variance during computing. Since the selection of parameters is completely random; and since no intelligence is used to sample these combinations, luck plays its part. (…) This works best under the assumption that not all hyperparameters are equally important. (…) The chances of finding the optimal parameter are comparatively higher in random search because of the random search pattern (…)".* **(Maladkar, 2018)**
- **XGBoost** is an implementation of gradient boosted decision trees designed for speed and performance. It is an open-source library and a part of the Distributed Machine Learning Community.

  As a newer and improved version of a gradient boosting algorithm it contains major enhancements compared with the other boosting methods, such as:

  1. **Regularization** comes in a handful of forms and in general it is meant to reduce the potential of overfitting prediction. It can also be used to perform feature selection.

There are two common types of regularization:

- **L1 Regularization (**Lasso) which pushes feature weights to zero and is represented by the alpha parameter.
- **L2 Regularization** (Ridge) which will push feature weights asymptotically to zero and is represented by the lambda parameter.

The use of which form of regularization is generally determined through cross-validation and parameter tuning.

2. Additionally, there is a **gamma parameter** which is meant to control the complexity of a given tree. The gamma parameter is used to set the minimum reduction in loss required to make a further split on a leaf node. XGBoost uses the gamma parameter in its pruning steps, grows the trees to a specified level and then uses the gamma parameter to determine which leaf nodes to remove.
3. The third enhancement is the **Parallelized development of the decision tree** ensembles. In this enhancement, the algorithm is not growing the decision trees in parallel but rather uses pre-sorting and block storage methods in parallel processing to determine what is the optimal split point. XGBoost will make use of a balance between in-memory and out-of-memory processes to make the most use of the computer's performance.
4. The last enhancement that deserves to be mentioned is the **Addition of drop out** to the boosting algorithm. Dropout is another type of regularization that is intended to reduce overfitting. Dropout is the process of removing randomly selected trees from the decision tree ensemble created in the model.

XGBoost contains a few parameters that are widely used and can be changed in order to improve the model performance and reduce overfitting, such as:

1. Number of Estimators – The number of boosting stages to perform, normally a larger number results in a better model performance.
2. Max depth – The maximum depth of a tree, normally increasing this value will make the model more complex and more likely to overfit.
3. Learning Rate – The learning rate shrinks the contribution of each tree on our ensemble. There is a trade-off between the Learning Rate and the Number of Estimators.
4. Columns Sample by Tree – This parameter represents the subsample ratio of columns (features) when constructing each tree. Subsampling occurs once for every tree constructed.
5. Subsampling - Subsample ratio of the training instances, when defined as 0.5, XGBoost randomly samples half of the training data prior to growing trees.
6. This parameter is prone to prevent overfitting. Subsampling occurs once in every boosting iteration.
7. Gamma – The game parameter represents the minimum loss reduction required to make a further partition on a leaf node of the tree.
8. Minimum Sum of Instance Weight needed in a child node - If the tree partition step results in a leaf node with the sum of instance weight less than the Minimum Sum of Instance Weight, then the building process will give up further partitioning.
9. Eta - Step size shrinkage used to prevent overfitting.

After each boosting step, it is possible to directly get the weights of the new features and shrink the features weights.

There are many more parameters regarding this ensemble algorithm, the list is very exhaustive so the ones that were mostly used on this report are the ones explained above.

- **Stratified K-fold** is an implementation of K-fold used for cross-validation. The splitting of data into folds follows the criteria in which each fold has the same proportion of observations with a given categorical value, such as the class outcome value.

III. METHODOLOGY

Since we want to predict the values of income, our target, we are dealing with a supervised learning problem. So, for the development of the project we will follow a standard approach for this type of classification problem. The key steps we will focus on will be the following:

1. Data access, exploration and understanding
2. Data preparation
3. Feature selection
4. Model selection
5. Improve Results
6. Final Results (Conclusion)

After we do a brief analysis and understanding of the data, we start by preparing it, this means separate the target from the rest of the features, and with that features, correcting inconsistencies, dealing with outliers, dropping useless features that won't help us predict our target, transforming features into more relevant ones, scaling the data, splitting the data into train and test, being the train part 70% of the data set and the test part 30% of the data set and finally filling missing values of each one. This step is fundamental to recognize and minimize any potential biases in our data sets, this will also help us have a well-prepared data so that the efficiency of the model used in the future improves.

With the clean data, it is time to choose the most important features to include in our train model because the irrelevant and unneeded attributes from data do not contribute to the accuracy of a predictive model or may in fact decrease it. To do that we use several methods including lasso regression, ridge regression, decision trees, recursive feature elimination, chi square test and ANOVA test.

After the feature selection, we are left with different groups of features resulting from the different methods used in feature selection. In model selection, before going any further, we apply a cross-validation technique to choose the best hyperparameters to each model. By applying this technique, we are able to boost the performance of our models and consequently the predictions. After that we will combine those different groups of features with several models in order to see which are the best ones and the best groups of features for each model. Knowing the best models to test, our purpose now is to improve the performance of each one, this will allow us to obtain a better score prediction. To do that we perform the hyperparameters tunning, changing the parameters and checking the scores for each change, we do that for the previous chose models. After the hyperparameters tunning, for each model, we also do a grid search to find the values that work best for our data set.

Finishing the results improvement, all the data transformations made to the train data, are applied on the new test dataset, in order to apply the resulting models to predict the target of the given test dataset and find the scores on Kaggle. At the end we will keep the model with the best score on Kaggle and the less overfitted scores on our Jupiter notebook (between train and test data).

IV. RESULTS AND DISCUSSION

1. *Prepare Data:*

Our original features

The first thing we did was to separate the target (y) from the rest of our data (X), which in our case was the Income feature.

Variable	Description
Citizen_ID	Unique identifier of the citizen
Name	Name of the citizen (First name and surname)
Birthday	The date of Birth
Native Continent	The continent where the citizen belong in the planet Earth
Marital Status	The marital status of the citizen
Lives with	The household environment of the citizen
Base Area	The neighborhood of the citizen in Newland
Education Level	The education level of the citizen
Years of Education	The number of years of education of the citizen
Employment Sector	The employment sector of the citizen
Role	The job role of the citizen
Working Hours per week	The number of working hours per week of the citizen
Money Received	The money payed to the elements of Group B
Ticket Price	The money received by the elements of Group C
Income	The dependent variable (Where 1 is Income higher than the average and 0 Income Lower or equal to the average)

FIGURE 1- ORIGINAL FEATURES

Feature engineering

Inconsistencies

- Birthday - we checked for inconsistencies and found people whose birthday were on 29th February in years that were not leap years. In order to correct these mistakes, we assumed that their birthday was on the 1st March. Also, we change the type to datetime.

New features

- Age – feature resulting from 'Birthday' year that relates to citizens age in the year of 2048.
- Gender – feature resulting from 'Name' title (Mr., Mrs., Miss) that relates to citizen's gender.

Transforming Feature values:

- Base area – we checked and the value 'Northbury' consisted in 89% of the feature values. The rest of the 11% were composed by 38 different values. So, we reduced this high cardinality into 2 different values (Northbury and Other Base Area).
- Marital Status – we simplified the values by combining all the 'Married' values and the 'Separated' ones.
- Education Level – we applied ordinal encoding to order the levels of education.

Dealing with outliers:

We started by visualizing the boxplots of each metric feature:

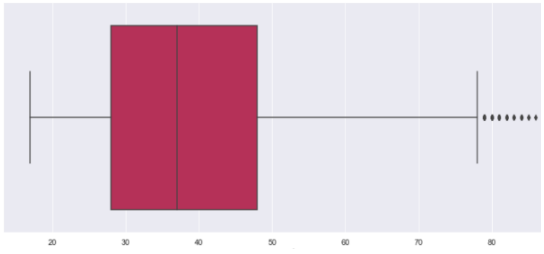


FIGURE 5 -AGE BOX PLOT

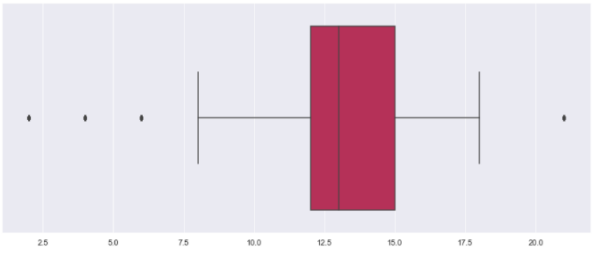


FIGURE 4- YEARS OF EDUCATION BOX PLOT

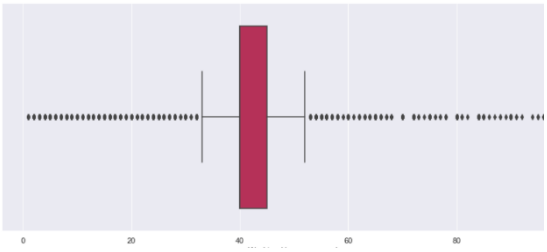


FIGURE 3- WORKING HOURS PER WEEK BOX PLOT

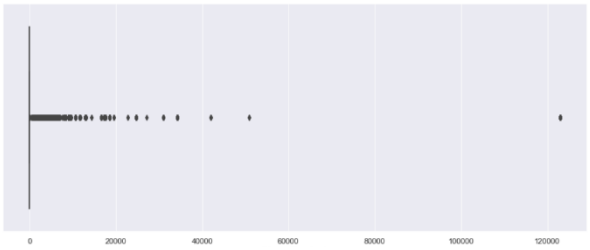


FIGURE 2- MONEY RECEIVED BOXPLOT

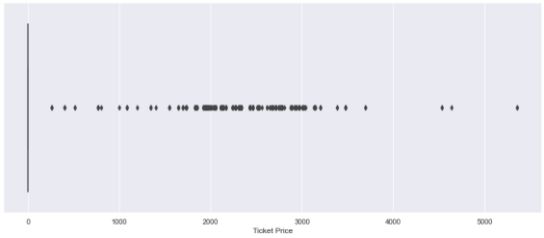


FIGURE 6 - TOTAL PRICE BOX PLOT

After we tried to deal with outliers using DBSCAN and LocalOutlierFactor, we concluded that, for this data, the best way was the manual way, based on the box plot observations, and our prior knowledge of what makes sense to be considered an outlier on demographic features. This decision was made because the percentage of data removed for both DBSCAN and LocalOutlierFactor was above 10%, and our purpose is not to remove more than 4% of data.

```
filters = ((data_original['Age'] > 85) | (data_original['Years of Education'] < 5)|
           ((data_original['Working Hours per week'] < 10) | (data_original['Working Hours per week'] > 90)) |
           (data_original['Money Received'] > 40000) | ((data_original['Ticket Price'] > 4000) |
           (data_original['Ticket Price'][data_original['Ticket Price'] > 0] < 700)))
data = data_original[~filters].copy()
target = target_original[~filters].copy()
```

```
print('Percentage of data kept after removing outliers:', np.round(data.shape[0]/data_original.shape[0], 4))
```

```
Percentage of data kept after removing outliers: 0.969
```

FIGURE 7- MANUAL FILTER METHOD

Split into train and test:

We slipped the data into train (70%) and test (30%), for both target (y) and the rest of the data (X), resulting in y_train, y_test, X_train and X_test.

Imputing the missing values:

The only feature containing missing values was “Role”, the one related to the job role of the citizen. Since we are dealing with a categorical feature, the method used to fill the missing values was the mode. At this point we have already split the data, so we filled the missing values on the X_train and on the X_test with the corresponding mode of each one for the “Role” feature.

Creation of dummies variables:

Native Continent, Lives with, Base Area, Employment Sector, Role, Marital Status are all categorical features, so we transform those into dummy variables to facilitate the application of the models. Then again, we did this for both X_train and X_test. This results in 46 total features.

Feature Scaling:

We used MinMaxScaler () to scale our features.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

FIGURE 8 - MINMAXSCALER () FORMULA

Correlation Analysis:

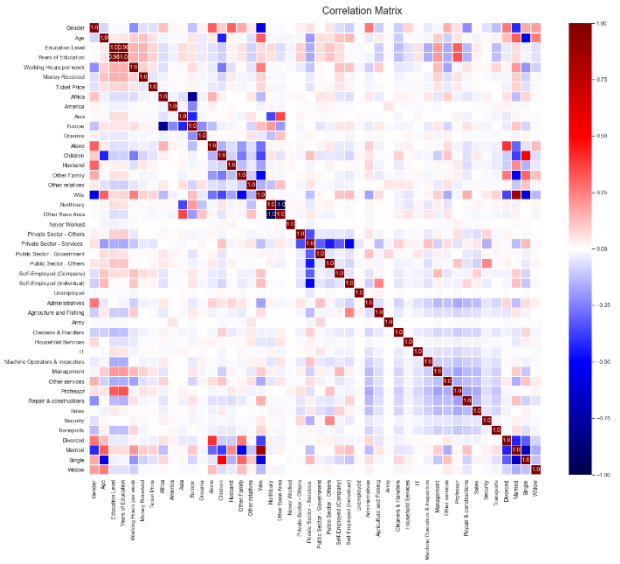


FIGURE 10- CORRELATION MATRIX FOR X_TRAIN

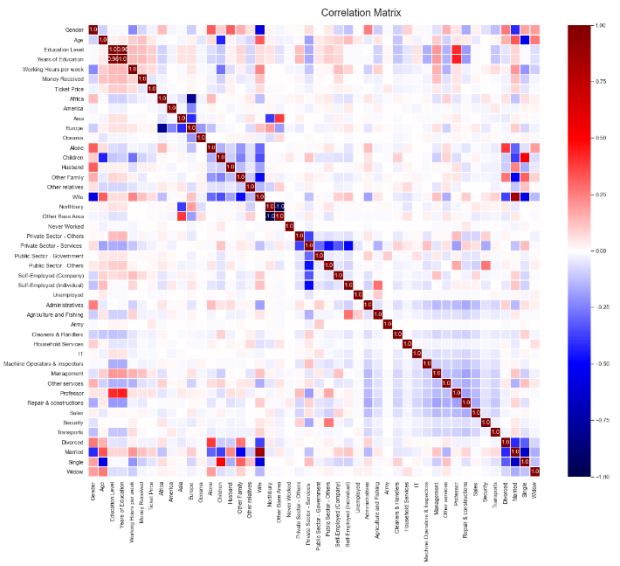


FIGURE 9 - CORRELATION MATRIX FOR X_TEST

We can see that the pair of features Education level and Years of Education and the pair Other Base Area and Northbury are both very high correlated, being the correlations 0.98 and -1 respectively. Because of that we removed one of each pair in order not to have redundant data. This means that we removed the features "Education Level" and "Other Base Area", from the train and test.

2. Feature Selection

- **Decision trees method:**

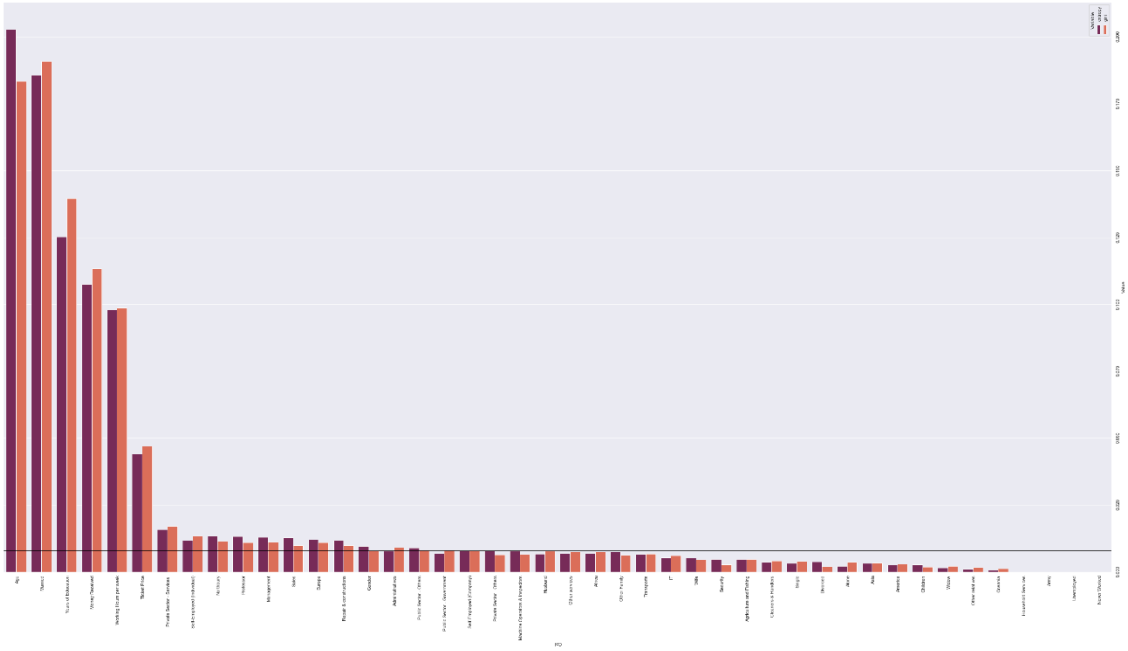


FIGURE 11 - GINI AND ENTROPY FEATURE IMPORTANCE

The important features are the ones above the line for entropy or gini metrics. That leaves us with the features for this method: Age, Europe, Management, Ticket Price, Gender, Professor, Northbury, Self-Employed (Individual), Money Received, Married, Sales, Administratives, Private Sector - Services, Years of Education, Public Sector - Others, Repair & constructions, Working Hours per week.

- **ANOVA Correlation Coefficient**

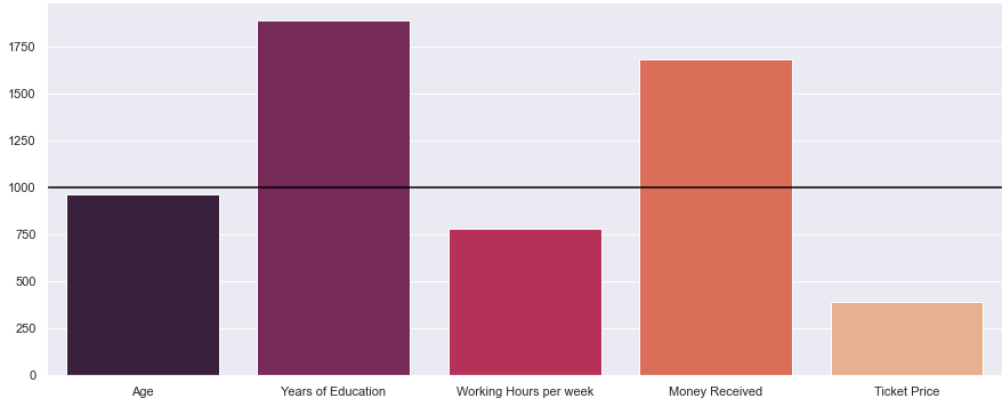
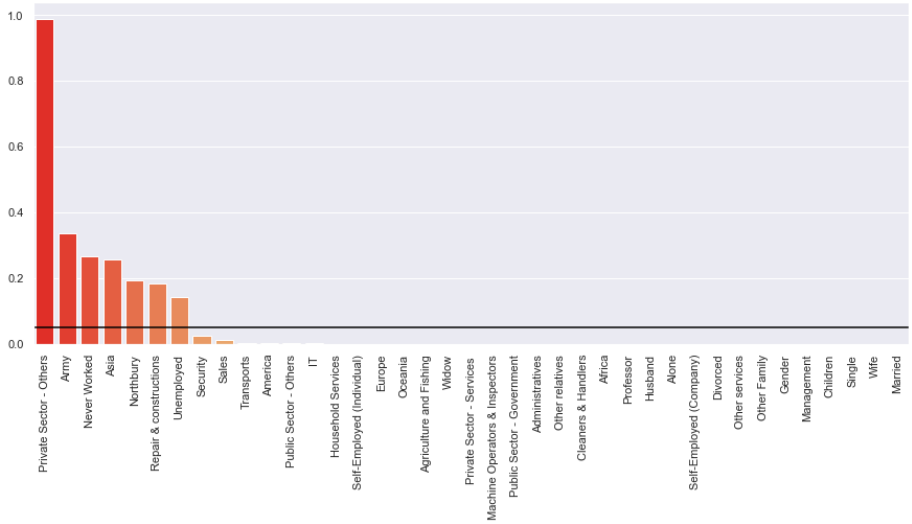


FIGURE 12 - NOMINAL DATA IMPORTANCE

This method tests only the importance of the nominal features. The important ones are the ones above the line, that represents the higher F statistics. So that leaves us with the features for this method: Years of Education and Money Received.

- **Chi Squared Test**

FIGURE 13 - CHI SQUARE FOR CATEGORICAL IMPORTANCE



This method tests only the importance of the categorical features. The important variables are the ones that have p-value bellow the line for 5% level of significance. So that leaves us with the features: Security, Transports, America, Sales, IT, Oceania, Private Sector - Services, Cleaners & Handlers, Machine Operators & Inspectors, Africa, Divorced, Europe, Other services, Professor, Administratives, Children, Self-Employed (Company), Other Family, Household Services, Gender, Single, Husband, Public Sector - Government, Widow, Management, Wife, Self-Employed (Individual), Other relatives, Married, Alone, Agriculture and Fishing, Public Sector - Others.

Once the last two methods were tested for different types of features, we combined the resulting features of each method. So, for the method Anova and chi square, the resulting features are: Security, Transports, America, Sales, IT, Oceania, Private Sector - Services , Cleaners & Handlers, Machine Operators & Inspectors, Africa, Divorced, Europe, Other services, Professor, Administratives, Children, Self-Employed (Company), Other Family, Household Services, Gender, Single, Husband, Public Sector - Government, Widow, Management, Wife, Self-Employed (Individual), Other relatives, Married, Alone, Agriculture and Fishing, Public Sector - Others, Years of Education, Money Received. We decided to call this group of features “Correlations”.

- **Ridge Regression method:**

The important features for the ridge regression method are the ones that have an absolute value of coefficient importance above the threshold defined with a red line. That leaves us with the features: Gender, Age, Years of Education, Working Hours per week, Money Received, Ticket Price, Alone, Children, Husband, Other Family, Other relatives, Wife, Private Sector - Others, Public Sector - Government, Self-

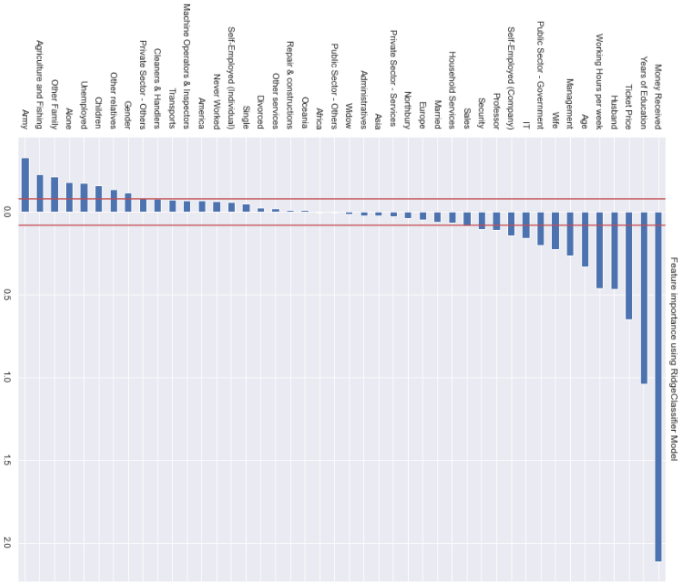


FIGURE 14 - FEATURE IMPORTANCE WITH RIDGE REGRESSION

Employed (Company), Unemployed, Agriculture and Fishing, Army, IT, Management, Professor, Sales, Security.

- **Lasso Regression method**

Like the previous one, the important features for the Lasso Regression method are the ones that are not between the lines representing the importance coefficient. That leaves us with the features: Age, Years of Education, Working Hours per week, Money Received, Ticket Price, Husband, Wife, Public Sector - Government, Agriculture and Fishing, Management.

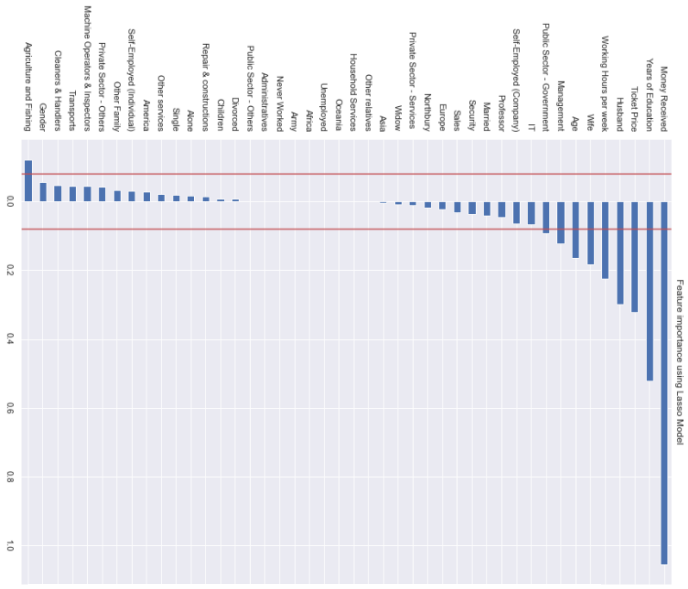


FIGURE 15 - FEATURE IMPORTANCE WITH LASSO REGRESSION

- **RFE (Recursive Feature Elimination) method**

Once RFE is a wrapper-method for feature selection, we used these other algorithms to be wrapped by RFE. This method will choose the features that provides the best score by eliminating the worse features for each iteration.

For Logistic Regression: leaving us with the features Gender, Age, Years of Education, Working Hours per week, Money Received, Ticket Price, America, Asia, Europe, Oceania, Alone, Children, Husband, Other Family, Other relatives, Wife, Northbury, Never Worked, Private Sector - Others, Public Sector - Government, Self-Employed (Company), Self-Employed (Individual), Unemployed, Administratives, Agriculture and Fishing, Army, Cleaners & Handlers, Household Services, IT, Machine Operators & Inspectors, Management, Other services, Professor, Repair & constructions, Sales, Security, Divorced, Married, Single, Widow.

For Decision Tress: leaving us with the features Years of Education, Money Received, Married.

For Gradient Boosting: leaving us with the features Gender, Age, Years of Education, Working Hours per week, Money Received, Ticket Price, Europe, Children, Husband, Wife, Northbury, Private Sector - Others, Private Sector - Services, Public Sector - Government, Self-Employed (Company), Self-Employed (Individual), Agriculture and Fishing, Cleaners & Handlers, IT, Machine Operators & Inspectors, Management, Other services, Professor, Sales, Transports, Married, Single, Widow.

For XGBoost: leaving us with the features Gender, Age, Years of Education, Working Hours per week, Money Received, Ticket Price, Africa, Asia, Europe, Alone, Children, Husband, Other Family, Other relatives, Wife, Northbury, Private Sector - Others, Private Sector - Services , Public Sector - Government, Public Sector - Others, Self-Employed (Individual), Administratives, Agriculture and Fishing, Cleaners & Handlers, IT, Machine Operators & Inspectors, Management, Other services, Professor, Repair & constructions, Sales, Security, Transports, Divorced, Married, Single, Widow.

3. Model Selection

The algorithm chosen for cross-validation was K-fold with ten splits. Before that, we tried to use Stratified K-fold due to the fact that we are dealing with an unbalanced dataset (76% zeros, 24% ones) but, since it was very time-consuming performing model assessment, we resigned to use this method.

For model selection we checked the following models:

Group of features / Models		Decision Trees	Correlations	Ridge Regression	Lasso Regression	RFE
Logistic Regression	**Train**	0.8450	0.8428	0.8466	0.8453	0.8495
	Val	0.8444	0.8421	0.8464	0.8451	0.8480
Neural Networks	**Train**	0.8611	0.8565	0.8631	0.8555	-
	Val	0.8494	0.8419	0.8516	0.8512	-
Decision Trees	**Train**	0.9767	0.8819	0.9639	0.9406	0.8470
	Val	0.8122	0.8351	0.8130	0.8253	0.8451
Naive Bayes	**Train**	0.8105	0.5872	0.5050	0.7950	-
	Val	0.8095	0.5862	0.5047	0.7949	-
KNN	**Train**	0.8761	0.8289	0.8784	0.8769	-
	Val	0.8208	0.8058	0.8337	0.8362	-
Ada Boost	**Train**	0.8551	0.8480	0.8581	0.8565	-
	Val	0.8545	0.8476	0.8562	0.8557	-
Gradient Boosting	**Train**	0.8632	0.8539	0.8673	0.8661	0.8670
	Val	0.8578	0.8501	0.8621	0.8616	0.8610
SVM	**Train**	0.8478	0.8460	0.8517	0.8510	-
	Val	0.8421	0.8403	0.8462	0.8485	-
XGBoost	**Train**	0.8980	0.8672	0.8978	0.8890	0.9048
	Val	0.8576	0.8555	0.8637	0.8611	0.8672

TABLE 1- MODEL SELECTION

These models were all built and tested with the parameters as default. We decided to follow three criteria to evaluate the best models:

- Highest number for score
- Lower Overfitting
- High score with high overfitting but with the potential to improve them later

As we can see the best models are marked down.

4. Improving Results

The models selected for hyperparameters tunning are: Neural Networks, Decision Trees, Ada Boost, Gradient Boosting and XGBoost. The groups of features that worked best for each model are:

- Neural Networks: Ridge Regression features
- Decision Trees: Decision Trees feature and Ridge Regression features
- Ada Boost: Ridge Regression features
- Gradient Boosting: Ridge Regression features and RFE features (for Gradient Boosting)
- XGBoost: Ridge Regression features and RFE features (for XGBoost)

For each one we did the hyperparameter tunning and chose the top best parameters to include in the Grid Search or in the Randomized Search in case the Grid Search is too much time consuming (days to run). These algorithms find the best combination of the previously chosen parameters to achieve the optimum score.

The best scores found for each one were as follow:

TABLE 2 - IMPROVED SCORES

Group of Features / Models		Decision Trees	Ridge Regression	RFE
Neural Networks (Grid Search)	**Solver 'lbfgs' Train**	-	0.8645	-
	Solver 'lbfgs' Test	-	0.8532	-
	Solver 'adam' Train	-	0.8650	-
	Solver 'adam' Test	-	0.8514	-
Decision Trees (Grid Search)	**Train**	0.8701	0.8745	-
	Test	0.8489	0.8517	-
Bagging for Decision Trees (Grid Search)	**Train**	0.8711	0.8713	-
	Test	0.8580	0.8570	-
Ada Boost (Grid Search)	**Train**	-	0.8686	-
	Test	-	0.8647	-
Gradient Boosting (Randomized Search)	**Train**	-	0.8809	0.8846
	Test	-	0.8633	0.8675
Gradient Boosting (Grid Search)	**Train**	-	-	0.8812
	Test	-	-	0.8686
XGBoost (Randomized Search)	**Train**	-	0.8753	0.8821
	Test	-	0.8639	0.8667

With the optimal parameters:

Neural Networks- Grid Search	**Ridge Variables** **Solver 'lbfgs'**	**Ridge Variables** **Solver 'adam'**
	{activation: 'logistic', hidden_layer_sizes: (11,), learning_rate_init: 0.0001, max_iter: 400, solver: 'lbfgs'}	{activation: 'relu', batch_size: 300, hidden_layer_sizes: (100,), learning_rate_init: 0.001, max_iter: 400, solver: 'adam'}
	Decision Trees	**Ridge Variables**
Decision Trees- Grid Search	{criterion: 'gini', max_depth: None, max_features: None, max_leaf_nodes: None, min_impurity_decrease: 0.0, min_samples_leaf: 1, min_samples_split: 100, min_weight_fraction_leaf: 0.0, random_state: 5, splitter: 'best'}	{criterion: 'entropy', max_depth: None, max_features: None, max_leaf_nodes: None, min_impurity_decrease: 0.0, min_samples_leaf: 1, min_samples_split: 100, min_weight_fraction_leaf: 0.0, random_state: 5, splitter: 'best'}
Bagging for Decision Trees- Grid Search	{base_estimator: DecisionTreeClassifier (min_samples_split=100, random_state=5), bootstrap: False, bootstrap_features: False, max_features: 0.75, max_samples: 0.75, n_estimators: 50, random_state: 5}	{base_estimator: DecisionTreeClassifier (criterion='entropy', min_samples_split=100, random_state=5), bootstrap: False, bootstrap_features: False, max_features: 0.75, max_samples: 0.5, n_estimators: 50, random_state: 5}
Ada Boost (Grid Search)	-	{algorithm: 'SAMME.R', learning_rate: 0.7000000000000001, n_estimators: 1000, random_state: 5}

TABLE 3 -IMPROVED PARAMETERS

	Ridge Variables	RFE
Gradient Boosting (Randomized Search)	{random_state: 5, n_estimators: 700, min_weight_fraction_leaf:0.005, min_samples_split: 10, min_samples_leaf: 50, min_impurity_decrease: 0, max_leaf_nodes: None, max_features: 'log2', max_depth: 3, loss: 'deviance', learning_rate: 0.3, criterion: 'mse'}	{random_state: 5, n_estimators: 100, min_weight_fraction_leaf: 0, min_samples_split: 10, min_samples_leaf: 50, min_impurity_decrease: 0.01, max_leaf_nodes: None, max_features: None, max_depth: 4, loss: 'deviance', learning_rate: 0.4, criterion: 'friedman_mse'}
Gradient Boosting (Grid Search)		{criterion: 'friedman_mse', learning_rate: 0.1, loss: 'deviance', max_depth: 5, max_features: None, max_leaf_nodes: None, min_impurity_decrease: 0.01, min_samples_leaf: 60, min_samples_split: 170, min_weight_fraction_leaf: 0, n_estimators: 250, random_state: 5}
XGBoost (Randomized Search)	{verbosity: 0, subsample: 1, random_state: 5, n_estimators: 200, min_child_weight: 1, max_depth: 4, learning_rate: 0.3, gamma: 3, eta: 0.05, colsample_bytree: 0.5}	{verbosity: 0, subsample: 0.5, random_state: 5, n_estimators: 100, min_child_weight: 1, max_depth: 5, learning_rate: 0.2, gamma: 3, eta: 0.07, colsample_bytree: 0.7}

TABLE 4 - IMPROVED PARAMETERS

As we can see in the table 2, the best scores emerged with the Gradient Boosting model and with the XGBoost model, stating this, we are going to discuss the results for those more specifically bellow.

Modelling with the Gradient Boosting after Randomized Search, we started to see interesting results to explore because the scores were high and even though the overfitting existed, it was possible to decrease it. Even though the Randomized Search is a "lighter" algorithm, it finds a solution **close** to the optimal if the hyperparameters contain the optimal solution. So being this model our best chance in order to find a better solution, we used the Grid Search on RFE variables with a smaller range of values for each parameter (showed on the table 4) to turn possible to run the code in a feasible time. To change the parameters to avoid overfitting, we chose smaller values for min_samples_split, min_samples_leaf, and learning_rate and higher

values for n_estimators. The given scores were not significantly different as we can see in table 2. For that, after research we found an implementation of the model Gradient Boosting called XGBoost. This implementation could help us to decrease the overfitting and get better results by converging faster to the optimal solution.

At the end, to test the capacity to decrease the overfitting even more, we applied the Bagging method (Stacking Classifier) with the models Gradient Boosting with the RFE features for Gradient Boosting, and XGBoost with RFE features for XGBoost. At the end with the merged models, we kept the features referring to RFE for XGBoost.

	RFE for XGBoost	
Stacking Classifier	value	0.8843
	test	0.8681

TABLE 5- STACKING CLASSIFIER

CONCLUSION

In conclusion, the model selected was the XGBoost with the following parameters:

- {verbosity: 0, subsample: 0.5, random_state: 5, n_estimators: 100, min_child_weight: 1, max_depth: 5, learning_rate: 0.2, gamma: 3, eta: 0.07, colsample_bytree: 0.7}

with the RFE features for XGBoost:

- *Gender, Age, Years of Education, Working Hours per week, Money Received, Ticket Price, Africa, Asia, Europe, Alone, Children, Husband, Other Family, Other relatives, Wife, Northbury, Private Sector - Others, Private Sector - Services , Public Sector - Government, Public Sector - Others, Self-Employed (Individual), Administratives, Agriculture and Fishing, Cleaners & Handlers, IT, Machine Operators & Inspectors, Management, Other services, Professor, Repair & constructions, Sales, Security, Transports, Divorced, Married, Single, Widow*

with the score of 0.8821 on train data, 0.8667 on test data, and 0.86831 on Kaggle.

This decision was made based on the higher and less overfitted score. We can see that the Stacking Classifier gave us a higher score, but the overfitting was higher too, and since the difference between scores is not significant, we kept the XGBoost one for the final result.

At the end we can conclude that 86.831% of the income is correctly predicted for 30% of the data.

REFERENCES

- Brownlee, Jason (February 10, 2019*).* ***Your First Machine Learning Project in Python Step-By-Step*** - https://machinelearningmastery.com/machine-learning-in-python-step-by-step/
- *(2020).* ***XBoost Parameters*** https://xgboost.readthedocs.io/en/latest/index.html/ (The author was not able to be found)
- S. Malik, R. Harod, A.S. Kunwar (February 3, 2020). ***XGBoost: A Deep Dive into Boosting***
- Swamynathan, Manohar (2019). ***Mastering Machine Learning with Python in Six Steps: A Practical Implementation Guide to Predictive Data Analytics Using Python, Second Edition***
- C. Müller, Andreas; Guido, Sarah (Jan 13, 2017)- ***Introduction to Machine Learning with Python: A Guide for Data Scientists***
- Brownlee, Jason (August 17, 2016*).* ***A Gentle Introduction to XGBoost for Applied Machine Learning*** - https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
- Brownlee, Jason (August 18, 2020*).* ***How to Perform Feature Selection with Numerical Input Data***- https://machinelearningmastery.com/feature-selection-with-numerical-input-data/
- ODSC – Open Data Science (December 27, 2019). **XGBoost: Enhancement Over Gradient Boosting Machines -** https://medium.com/@ODSC/xgboost-enhancement-over-gradient-boosting-machines-73abafa49b14/
- ***Machine Learning's Practical and theoretical classes***
- ***Scikit learn library Documentation***

- Quotes

I. Maladkar, K. (2018, 06 14). **Why Is Random Search Better Than Grid Search For Machine Learning.**