Newland: A Machine Learning Approach

"Predicting taxpayer groups for a new society"

Leonardo Freitas (m20200407@novaims.unl.pt), Pedro Sancho Vivas de Castro (m20200132@novaims.unl.pt), Vilmar Adriano Bussolaro (m20200268@novaims.unl.pt), Xavier Golaio Gonçalves (m20201090@novaims.unl.pt)

I. INTRODUCTION

In this practical group assignment for Machine Learning course, we have been asked to work as Data Scientists to create a predictive model to correctly identify individuals´ tax brackets into a fictional society in the future, called "Newland".

The competition was posted on Kaggle, a well-known ML contest platform. On the website we found an outline of the problem as well as two datasets in .csv files: Train and Test.

Both files have the same number of *features* (independent variables), but "Train.csv" has the outcome of the *prediction* (dependent variable defined by 0 or 1). While "Test.csv" has all the *features* but no *prediction*. That missing prediction is the output of the model developed in this paper, which will be submitted on the Kaggle competition.

Let´s start with the problem. In the year 2048, the newly discovered young planet of Newland has been gradually populated. The local government faces a budget constraint to finance public services and needs to implement a tax system from scratch. Thus, the inhabitants of the new planet will start to pay progressive[1] income taxes. Policy makers decided to create two income brackets, applying a 15% income tax rate to people (over 17) with an income below or equal to the average income of the entire population. The rest of the taxpayers must pay 30% of their income.

The question posed by Newland´s government is to identify the people who belong to each class, using machine learning predictive models. Correctly identifying people prior to their arrival to the new planet is important to support the local government on better planning its budget allocation and policies.

Our team received two datasets: "Train.csv" has 22.400 observations - a training set with the ground truth associated to each citizen, indicating to what group the citizen is related, regarding their income. The training set will be used to build our machine learning models and assess the performance. The "Test.csv" has 10.100 individuals. By applying the model previously built, we can get some estimation of the performance on unseen data by submitting the predictions to Kaggle, where 30% of the test data is assessed according to the F1 Score.

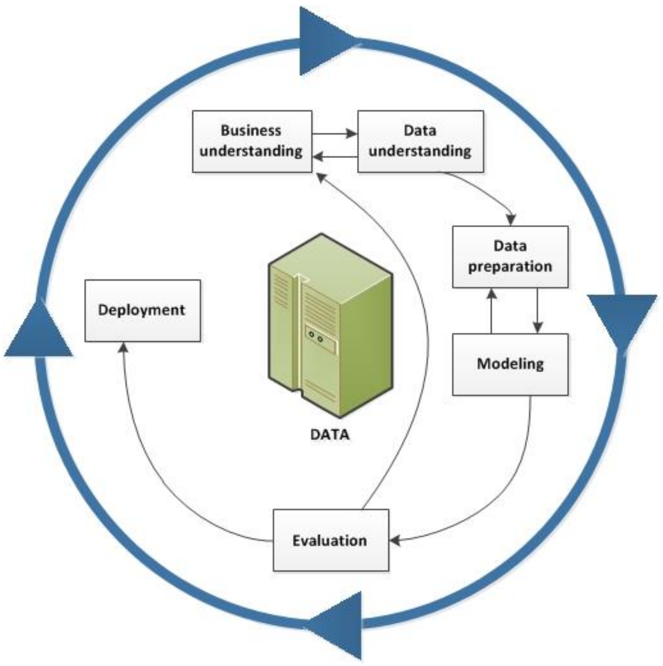
[1] "The definition of a progressive tax system usually starts with the idea of a proportional tax, in which everyone pays the same share of income in taxes. From that baseline, a progressive tax is one in which the share of income paid in taxes rises with income, and a regressive tax is one in which the share of income paid in taxes falls with income. Of course, real-world tax codes are complex and full of rules that have different effects across the income distribution. Thus, a more general definition is that a tax system can be defined as progressive if after-tax income is more equally distributed than before-tax income, and regressive if after-tax income is less equally distributed than before-tax income. " *(How Progressive is the U.S. Federal Tax System? A Historical and International Perspective* - Journal of Economic Perspectives—Volume 21, Number 1—Winter 2006—Pages 000–000 - *Thomas Piketty and Emmanuel Saez)*

The F-score, also called the F1-score, is a measure of a model's accuracy on a dataset. It is used to evaluate binary classification systems, which classifies examples into 'positive' or 'negative'. The F-score is a way of combining the precision and recall of the model, and it is defined as the harmonic mean of the model's precision and recall. The formula for the standard F1-score is the harmonic mean of the precision and recall. A perfect model has an F-score of 1 (Wood, 2020).

II. METHODOLOGY

To tackle the problem proposed, we will build upon a simplified version of the CRISP-DM framework recommended by Kelleher, Mac Namee, & D'arcy (2015, p. 53), from the textbook of this discipline "Fundamentals of Machine Learning for Predictive Data Analytics".

As a process, the life cycle model consists of six phases with arrows indicating the most important and frequent dependencies between phases. The sequence of the phases is not strict. In fact, most projects move back and forth between phases, as necessary. The figure emphasizes that data is at the heart of the process. Certain phases in CRISP-DM are more closely linked together than others. For example, Data Preparation and Modeling phases are closely linked, and analytics projects often spend some time iterating between these two phases. (Kelleher, Mac Namee & D´arcy, 2015, p. 54).



2

The logic behind our work can be summarized in the six key phases cited above. But this specific group project will take more time and effort on the final steps.

[2] The data mining life cycle. Image from IBM Knowledge Center, https://www.ibm.com/support/knowledgecenter/it/SS3RA7_sub/modeler_crispdm_ddita/clementine/crisp_help/crisp_overview.html

III. Building upon the framework

Business and Data Understanding

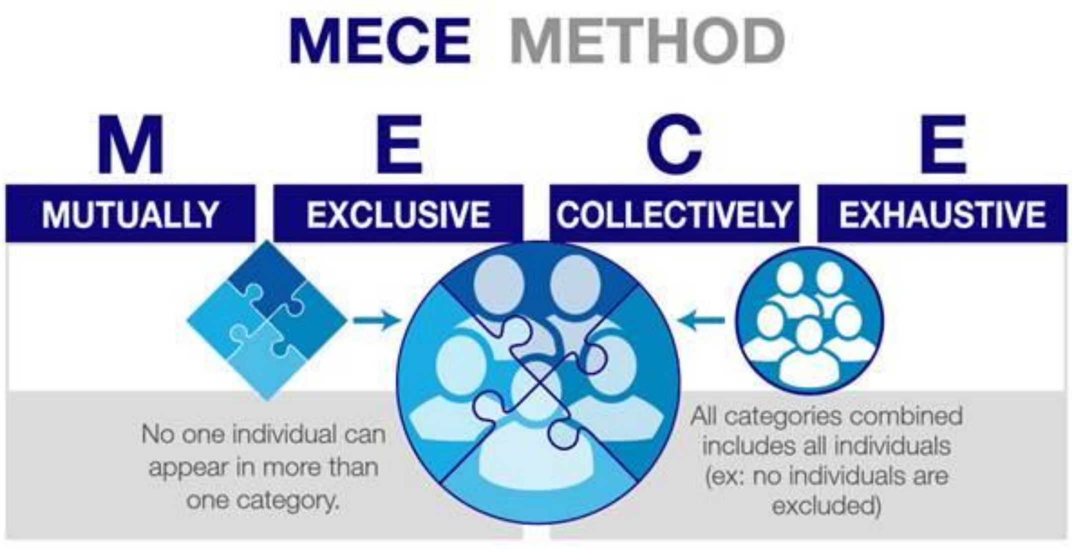
Steps 1 and 2, are tightly coupled together. These steps have been worked out in the Introduction and were made simple by the instructors of the discipline: we don´t have missing values in our dataset, our work structure and objectives were clear and concise, so we have the chance to work on matters more pertinent to the subject: Data Preparation, Modeling and Evaluation. Those will be covered on the next chapter.

Data Preparation

Despite not having a lot of steps to do in Data Preparation, since the data was fairly organized, we did try several approaches with Feature Selection and Feature Engineering.

Thematical feature engineering and Decision Trees

Our first approach was using a Mutually Excluding Complementary Exhaustive method (in a thematical fashion) to create some dummy variables but also trying to avoid too many features that could have extremely imbalanced histograms. The logic behind that was to extract information out of categorical data we had without converging to a solution that could be easily implemented with OneHotEncoding Sklearn’s library. Therefore, this last could be a framework of a solution that was computationally more daunting but that generated less strain on coding and less need of a rationale behind on its decisions.


MECE METHOD
M
MUTUALLY
E
EXCLUSIVE
C
COLLECTIVELY
E
EXHAUSTIVE
No one individual can appear in more than one category.
All categories combined includes all individuals (ex: no individuals are excluded)

3

DATA PREPROCESSING - The goal was to be thematic, and in that sense create categorical variables using social variables, educational ones, gender, but always trying to avoid having 10 features with super imbalanced records between the two labels (1s and 0s). To avoid that issue many aggregations were done in original categorical variables such as Base Area and Native Continent. Since the first was the extreme case of having one Base that had almost all the records and having more than 10 bases with just a few records, it would not make sense to have a number of features extremely imbalanced on this approach, in fact only one binary variable (1-Northbury, 0-Non-northbury).

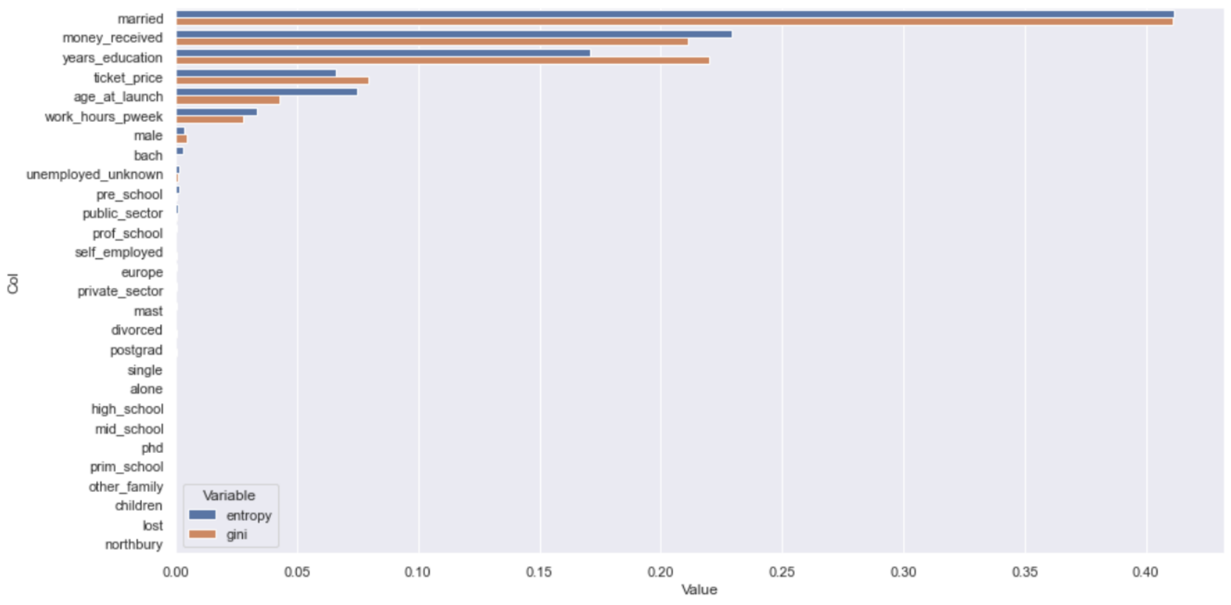
RFE RECURSIVE FEATURE ENGENEERING
RFE is popular because it is easy to configure and use and because it is effective at selecting those features (columns) in a training dataset that are more or most relevant in predicting the target variable.
There are two important configuration options when using RFE: the choice in the number of features to select and the choice of the algorithm used to help choose features. Both of these hyperparameters can be explored, although the performance of the method is not strongly dependent on these hyperparameters being configured well

Naturally, it would also be needed to proceed to create as many specifications as possible to evaluate which list of variables would provide more explanatory power, without running into the overfitting problem. Through implementing one of the most frequent used machine learning algorithms (Wu et al., 2007), the intention was to be able to generate a feature importance graph to assess and best showcase a clear decision rule for the specification of features implemented as well as to assess whether larger feature set was generating more predictive power. The main reasons why we choose the Decision Tree to do this first assessment of our Feature Engineering and to decide on our Feature Selection was applicability to all features already transformed, less demand on pre-processing, even scalability and interpretability of this particular algorithm. Lastly, since there was no outlier removal treatment done until this phase, among all classification algorithms Decision Trees was our decision given that is one of the most robust algorithms to outliers (Hastie et al., 2005).

Another way of assessing if the list of features were naturally important was to identify a number of list of variables to implement the algorithm and evaluate the spent time to run the train-test procedure assessment and the different F1-Scores and therefore continue to draw conclusions on the importance of specific variables to generate better prediction without unnecessary strain to compute the predictions.

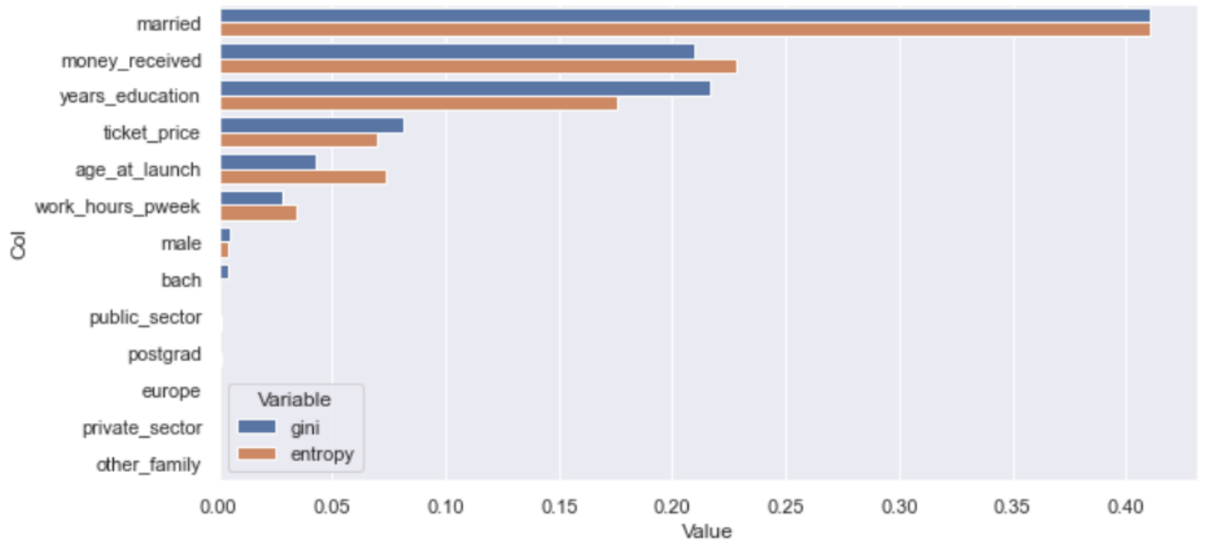
The diagnostic that continued to appear was that having an exhaustive feature set, e.g., as many thematic groups of features on our implementation was best at predicting using the train-test-split methodology. Additionally, we gravitated towards implementing the OneHotEncoding since it was an even more exhaustive approach on the features than having all the thematic group of features, we created on the first approach.

ONEHOT ENCODING A one hot encoding allows the representation of categorical data to be more expressive. Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers. This is required for both input and output variables that are categorical.



4

From the graph above, we can see the effect each feature has on *entropy* and *gini* (using Decision Tree Classifier). Above, a close-up of the most important features on the same specification of the previous graph but with only these features that explained most of the dataset information.



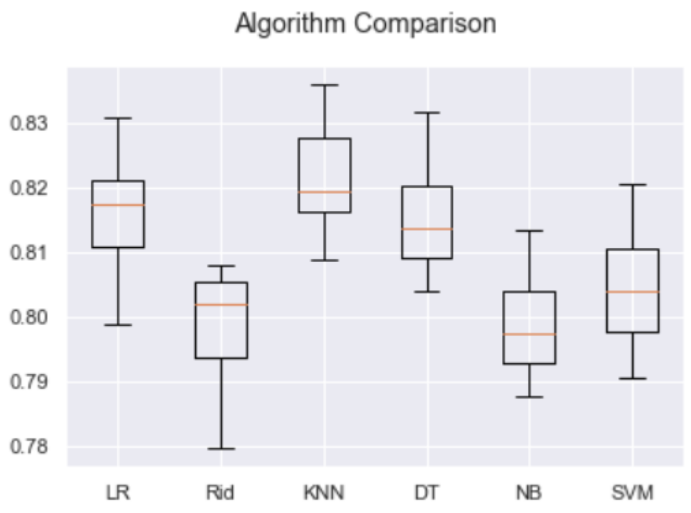
5

Although we converged to both treating some variables to create age and gender features and assessing the other categorical variables with OneHotEncoding, in order to gather more insight on the dataset we generated and assessed which features would be more important on the first approach through the least overfitted Decision Tree implementation, using *gini* and *entropy* (as shown before).

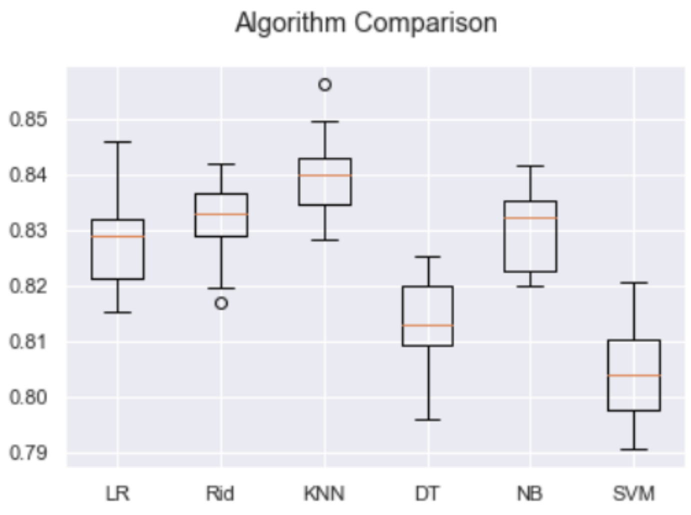
Through the last visualization, it seemed clear that even these few features could be more trimmed in a shorter feature set for a final decision on the features selected by the first approach, but in order to keep predictive power and be more parsimonious in eliminating data to generate more efficient implementations these few features seemed the right feature set.

10-fold Cross-validated assessment of thematical feature engineering

Our last effort to assess if more thematic variables and more features overall drawn from the categorical original variables were adding information to our implementation, we generated a number of box-plot implementation for only the metric features. Then, we proceed to do the same for a number of different list of variables that included different subsets of thematical dummy variables. The results can be seen in the next two graphs, showing the most exclusive one subset of features and the most inclusive one:



6

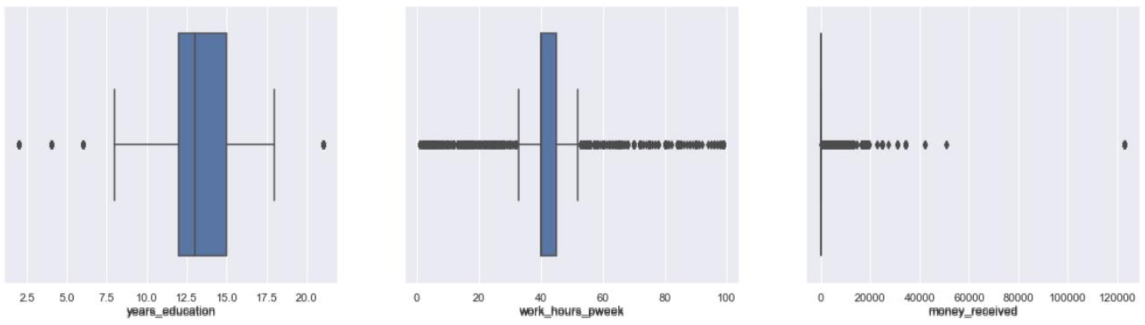


7

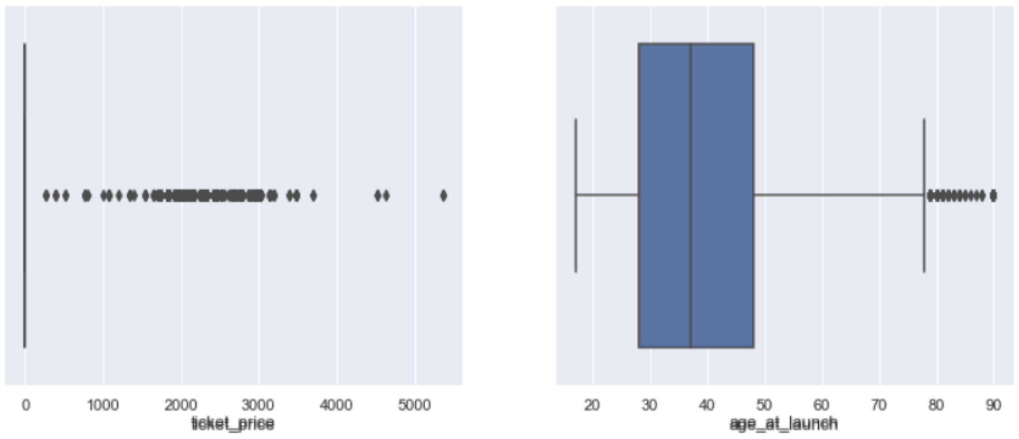
Using a few standard classification methods, most of the algorithms display better results using a K-fold CV implementation approach at the most inclusive one, and when diagnosing the spent time, it does not seem that difference is that daunting for this particular dataset. Therefore, with this last diagnostic picture we proceed to the second approach using mostly *OneHotEncoding* for our feature set.

Univariate Outlier Removal Assessment

Through generating box-plot on the metric features will be able to assess whether there are so-called univariate outliers or if there are none.

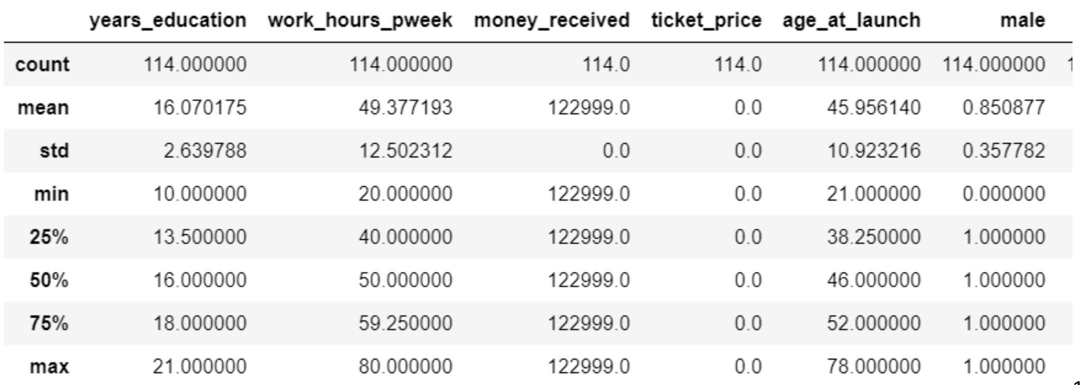


11



12

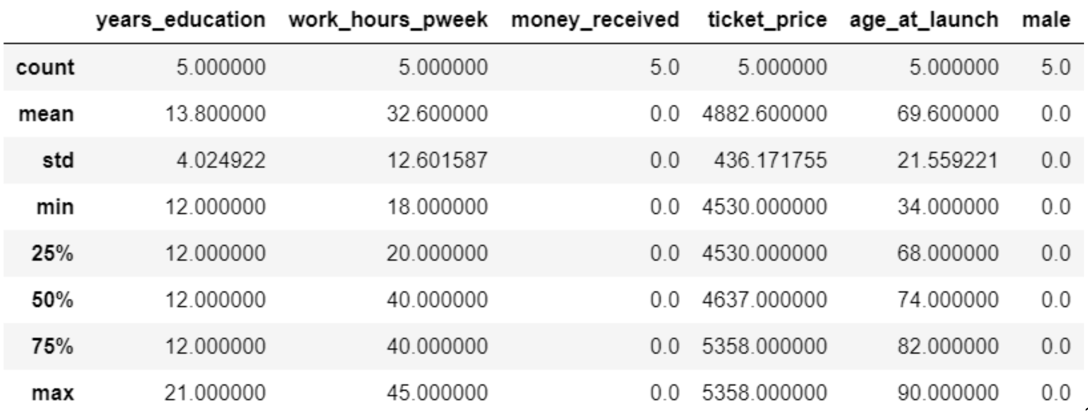
Following on that notion of being as parsimonious as possible with our selection of procedures I would say that, given that the distributions are non-normal, I would only consider for supposed univariate outliers records that had ticket price higher than 4000 and those that had money received higher than 60000. We will assess the size of this subsets and then proceed to analyze the F1-Score in a 10-fold CV fashion on multiple simple algorithms without changing arguments on each and every algorithm to assess whether we are improving the ability to generalize to the whole population by doing this more conservative approach on doing univariate outlier removal.



	years_education	work_hours_pweek	money_received	ticket_price	age_at_launch	male
count	114.000000	114.000000	114.0	114.0	114.000000	114.000000
mean	16.070175	49.377193	122999.0	0.0	45.956140	0.850877
std	2.639788	12.502312	0.0	0.0	10.923216	0.357782
min	10.000000	20.000000	122999.0	0.0	21.000000	0.000000
25%	13.500000	40.000000	122999.0	0.0	38.250000	1.000000
50%	16.000000	50.000000	122999.0	0.0	46.000000	1.000000
75%	18.000000	59.250000	122999.0	0.0	52.000000	1.000000
max	21.000000	80.000000	122999.0	0.0	78.000000	1.000000

13

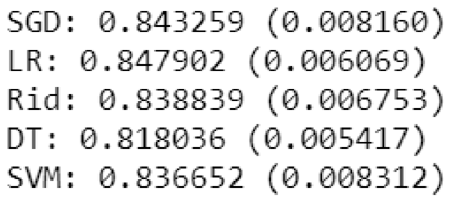
The profile of these subset of records does give us a hint that they are not outliers, despite having as little as 114 records that are separated with this cut-off in money received feature. The distribution of this subset is degenerated at 122999 mean, with zero standard deviation, which is probably something that will account for a small subset since is so concentrated in a single value, but probably not an outlier that should be out of the specification training. Another indication of that, is the high concentration on a subset of a common population that can be visualized through high concentration of males with high level of schooling.



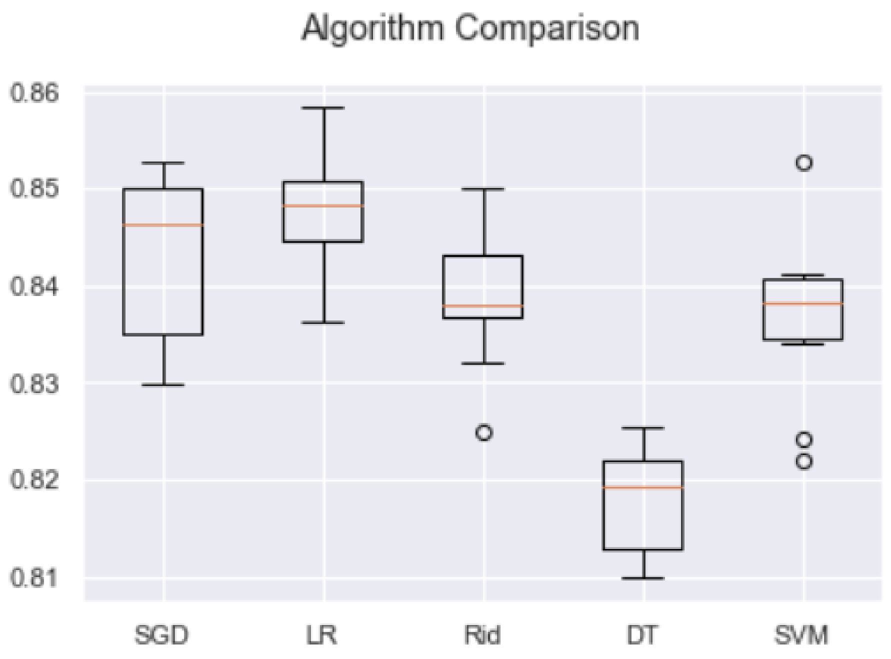
	years_education	work_hours_pweek	money_received	ticket_price	age_at_launch	male
count	5.000000	5.000000	5.0	5.000000	5.000000	5.0
mean	13.800000	32.600000	0.0	4882.600000	69.600000	0.0
std	4.024922	12.601587	0.0	436.171755	21.559221	0.0
min	12.000000	18.000000	0.0	4530.000000	34.000000	0.0
25%	12.000000	20.000000	0.0	4530.000000	68.000000	0.0
50%	12.000000	40.000000	0.0	4637.000000	74.000000	0.0
75%	12.000000	40.000000	0.0	5358.000000	82.000000	0.0
max	21.000000	45.000000	0.0	5358.000000	90.000000	0.0

14

Once again, the profile of these subset of records does give us a hint that they are not outliers, despite having as little as 5 records that are separated with this cut-off in ticket priced feature. The distribution of this subset is presenting only women, highly concentrated on the 12-year mark of education, probably a subset of the population that needs to be generalized. Nevertheless, we will proceed to do the 10-fold cross validated assessment of different algorithm implementations with both subsets of data and without the so-called univariate outliers. By using pipeling and a for loop we are to implement this procedure and generate the following results for the whole dataset and the dataset without the outliers, respectively:

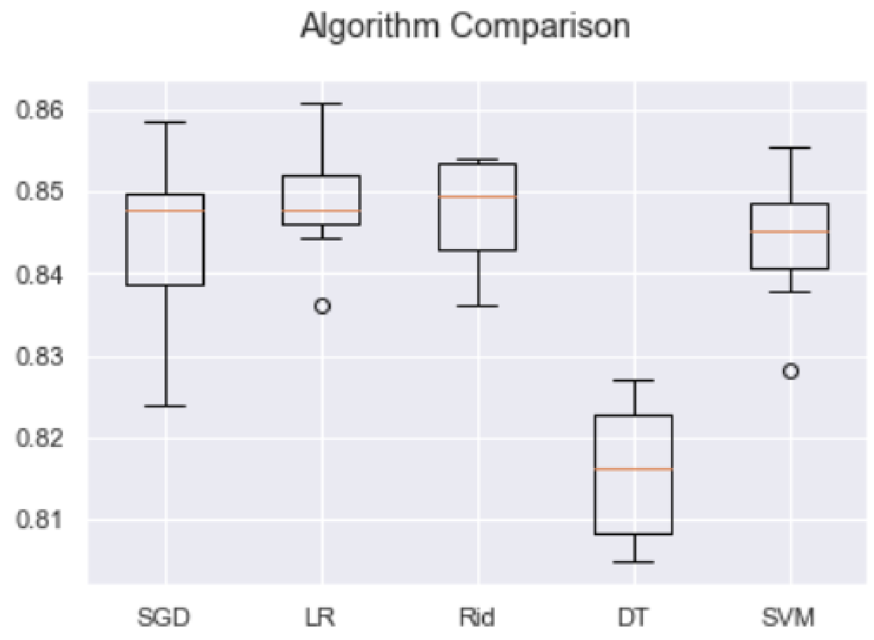


```
SGD: 0.843259 (0.008160)
LR: 0.847902 (0.006069)
Rid: 0.838839 (0.006753)
DT: 0.818036 (0.005417)
SVM: 0.836652 (0.008312)
```



15

```
SGD: 0.844397 (0.009251)
LR: 0.848750 (0.006450)
Rid: 0.847628 (0.006244)
DT: 0.815763 (0.007663)
SVM: 0.844127 (0.007256)
```

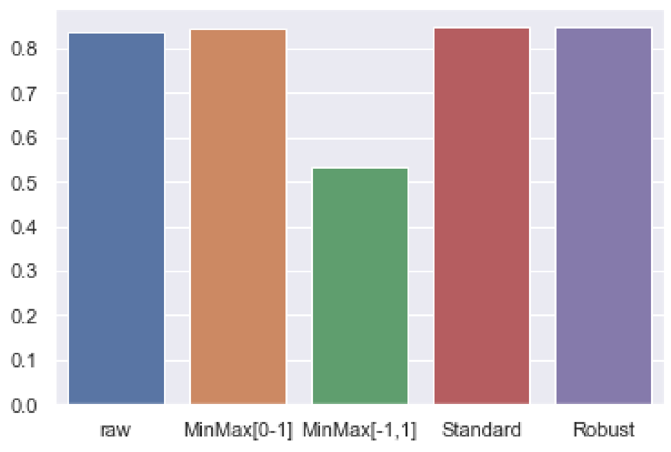


16

Since we have almost indistinguishable results for 2 algorithms (Decision Trees and Stochastic Gradient Descent), 1 algorithm that got slightly worse (Linear Regression) and 2 algorithms that got slightly better (Support Vector Machines and Ridge). Overall, standard deviations changed on similar profile thus strengthening our position to keep all the records and avoid the univariate outlier removal.

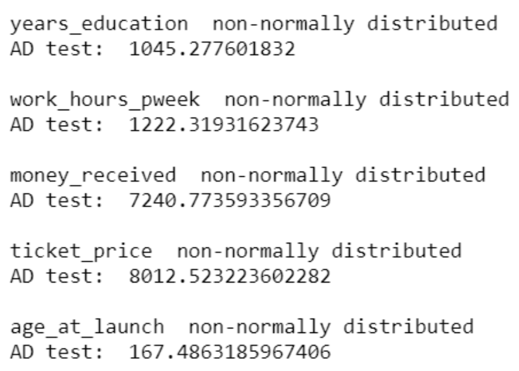
Scaling assessment and normality tests of metric features

Various scalers were used, such as: StandardScaler, RobustScaler, MinMaxScaler. Since MinMaxScaler does not assumes any distribution when doing the scaling and the difference in performance is not significant at first it seems more parsimonious to use MinMaxScaler.



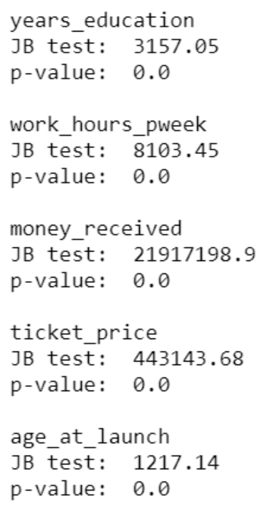
8

In order to further assess if using StandardScaler would be a good assumption on the distribution of the metric features we proceed to do two normality tests using the Scipy.stats implementation, such as the Jarque-Bera test and Anderson-Darling test (D’Agostino, 1986). The selection of this two normality test of a whole spectrum of viable tests is explained by adequateness in different aspects of each test. Jarque-Bera test will address better the skewness and kurtosis aspect of each feature comparing to the normal distribution and will yield more robust results when dealing with large quantities of records in a feature (Corder et al., 2011). On the other hand, Anderson-Darling test is more versatile to implement against any well-known distribution (the default argument is the normal distribution) and uses an empirically calculated cumulative density function to compute its test statistics and a common competitor for AD test, Shapiro-Wilk, will not compute very trustworthy test statistics will large datasets. Therefore, the joint information of these two normality tests in a single direction, which is, being normal or non-normal will clearly draw a marker on using MinMax[0-1] or the StandardScaler.



```
years_education   non-normally distributed
AD test:   1045.277601832

work_hours_pweek   non-normally distributed
AD test:   1222.31931623743

money_received   non-normally distributed
AD test:   7240.773593356709

ticket_price   non-normally distributed
AD test:   8012.523223602282

age_at_launch   non-normally distributed
AD test:   167.4863185967406
```

9



```
years_education
JB test:   3157.05
p-value:   0.0

work_hours_pweek
JB test:   8103.45
p-value:   0.0

money_received
JB test:   21917198.9
p-value:   0.0

ticket_price
JB test:   443143.68
p-value:   0.0

age_at_launch
JB test:   1217.14
p-value:   0.0
```

10

Next, we will proceed with using the best assumption that could be used as far as scaling by using MinMaxScaler, and using the [0-1] since it yielded the best results regarding the F1-Score.

Multivariate Outlier Removal Assessment

When analyzing the univariate outliers in the dataset something naturally arises from the end of that discussion, how will a multivariate outlier removal assessment will give to a few of the best model possible implementations (found through GridSearchCV implementation).

Using standard sklearn libraries, there was a few possible implementations that would make sense to implement in a 10-fold cross-validated fashion if at least showed any improvement on Mean Absolute Error metric. Therefore, we experimented with different arguments for OneClassSVM, LocalOutlierFactor, and IsolationForest as methods to extract multivariate supposed outliers from the whole dataset. The MAE metric showed small improvement on all of them, even when being very parsimonious when choosing the arguments of each method (at most taking out 1% of the dataset).

OneClassSVM method to identify multivariate outliers

Unsupervised outlier detection method that basically creates a high-dimension distribution will assign as outliers records that have very low probability of belonging to the high-dimensional distribution created (Lamrini et al., 2018). Contrary to Support Vector Machines in general, OneClassSVM creates a boundary that aims to achieve separation between the samples of the dataset and the origin. Given that only a small subset of dataset will lie on the other side of the boundary, which are the records considered as outliers by this method (Schölkopf et al., 2011). Since this method is generally more sensitive to outliers and we did not remove univariate outliers, it will probably not show increase in performance other than the MAE metric.

LocalOutlierFactor method to identify multivariate outliers

Another great implementation to remove outliers is the Local Outlier Factor (LOF) algorithm. It is an unsupervised anomaly detection method which computes the local density deviation of a given data point with respect to its neighbors. It considers as outliers the samples that have a substantially lower density than their neighbors. (Scikit Learn, 2020)

IsolationForest method to identify multivariate outliers

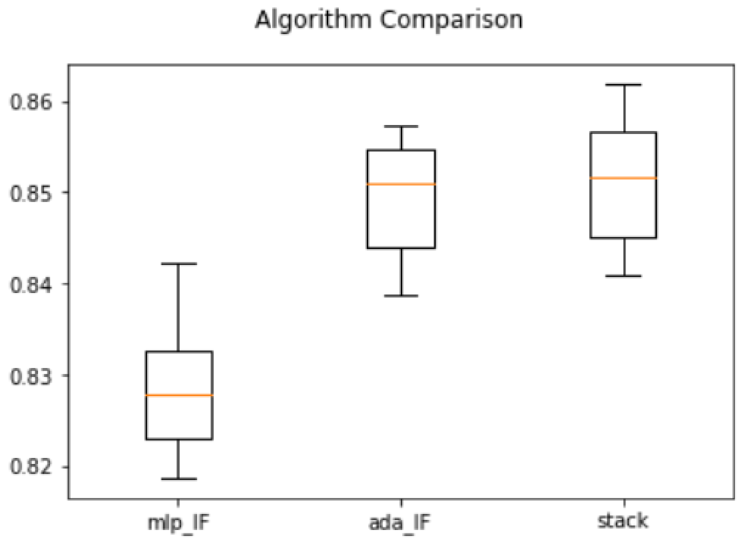
Lastly, a viable solution for analyzing outliers on the dataset is by using the standard sklearn IsolationForest to use the power of Random Forests. This procedure 'isolates' records by bootstrapping for a single feature and then randomly selecting a split value for the randomly drawn feature.

By bootstrapping the features when we have an outlier, we will naturally generate shorter paths. Therefore, when a forest of random trees collectively produces shorter path lengths (which is our measure of normality) for specific records reiteratively, this procedure will conclude that these specific records are in fact outliers in a multivariate fashion analysis (Liu et al., 2008). The same author states that it utilizes no distance or density measures to detect anomalies which eliminates major computational cost. Besides, it has the capacity to scale up to handle extremely large data size and high-dimensional problems with a large number of irrelevant attributes.

10-fold cross-validated assessment on multivariate outlier removal on F1-Score

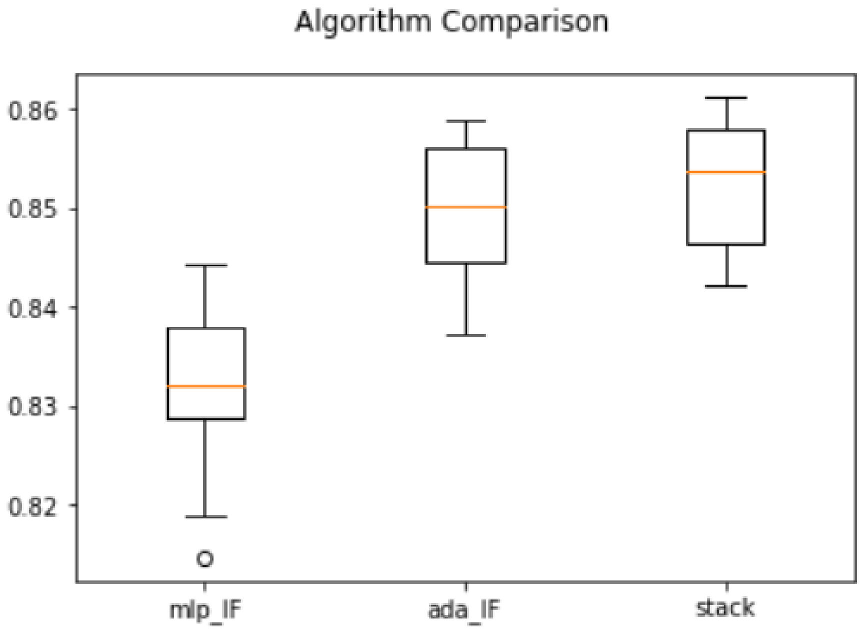
Naturally, there was the progression to see if it would check the F1-Score improvement (cross-validated mean value and its standard deviation), and we will proceed to show the results for OneClassSVM, Elliptic Envelope, LocalOutlierFactor and IsolationForest methods and the whole datasets with the so-called multivariate outliers, respectively:

```
mlp_IF: 0.828583 (0.007088)
ada_IF: 0.849220 (0.006298)
stack: 0.851357 (0.007104)
```



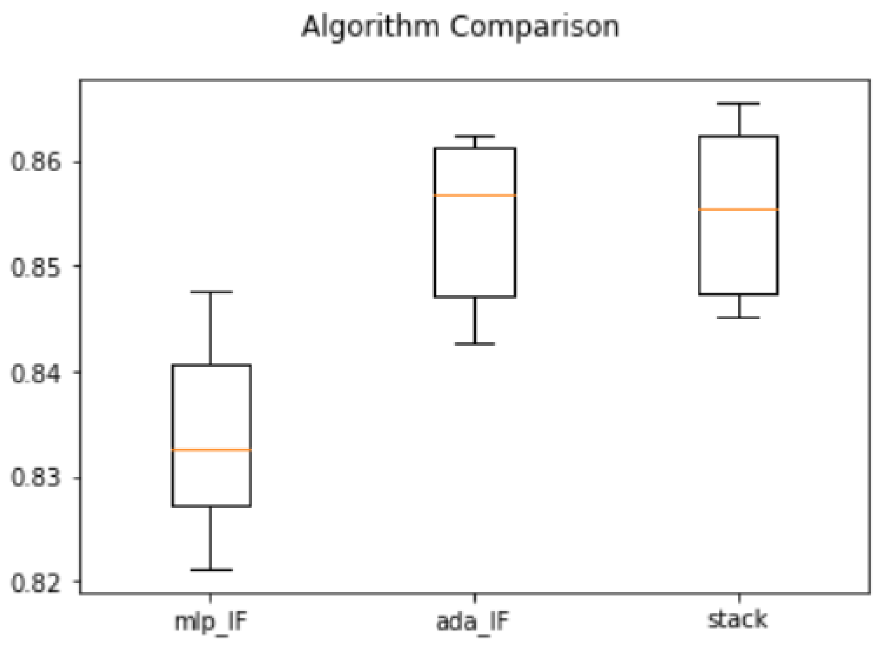
11

```
mlp_IF: 0.831394 (0.008713)
ada_IF: 0.849658 (0.007122)
stack: 0.852364 (0.006838)
```



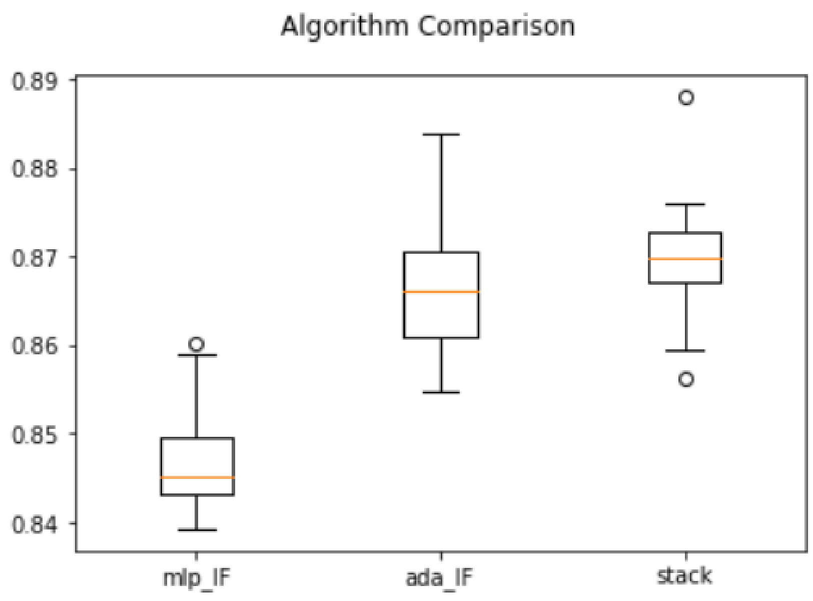
12

```
mlp_IF: 0.833733 (0.008183)
ada_IF: 0.854319 (0.007311)
stack: 0.855033 (0.007476)
```



13

```
mlp_IF: 0.847455 (0.006832)
ada_IF: 0.866429 (0.008131)
stack: 0.869911 (0.008290)
```



14

Using all the dataset we have better prediction and almost indifferent standard deviation using a robust method to assess through 3 algorithms: 2 algorithms found through GridSearchCV (Multi-layer Perceptron and AdaBoost) and a stacking implementation that generated better results than these two pure algorithms.

Additionally, this last graph also shows that stacking solutions are yielding better results than non-stacking implementations and therefore we will proceed into this type of diagnostics further ahead.

IV. Modeling and Evaluation:

Since we had our dataset ready, scaled and organized, we were ready to start testing. Our approach was to test the standard implementation of all classification algorithms available on Scikit-Learn, see the promising ones, and move forward with GridSearchCV to fine tune the best models according to our data.

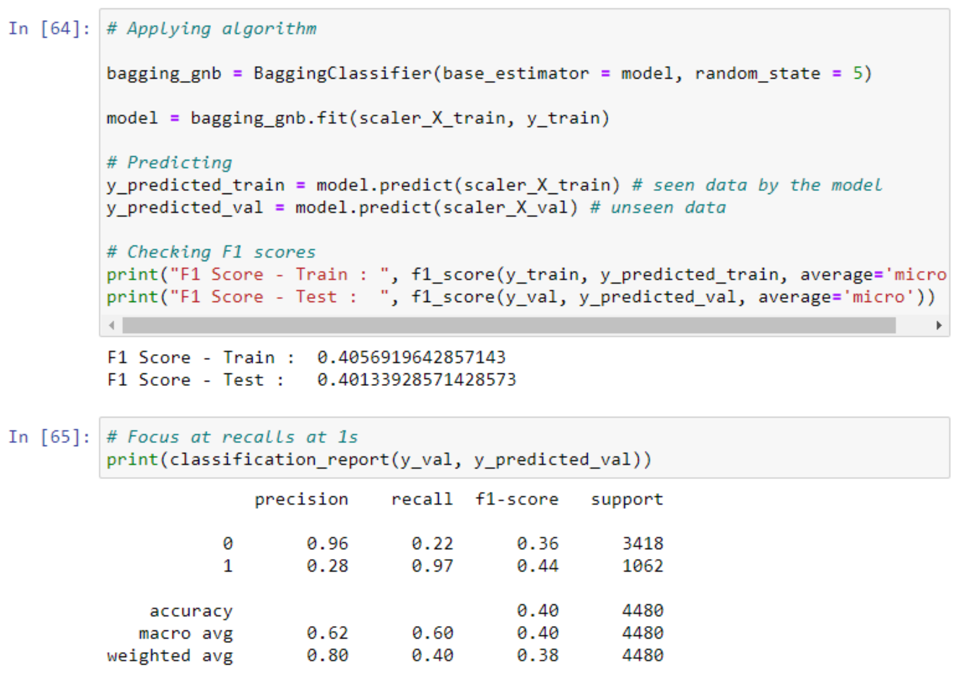
HYPERPARAMETER In machine learning, a hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are derived via training. ... Examples of algorithm hyperparameters are learning rate and mini-batch size

GridSearchCV algorithm was used to find the best hyperparameters to each algorithm that maximizes the validation score. Using “brute force”, this algorithm tries many different model parameters (taken from the algorithm documentation), simulate all possible combinations, at each grid point, on each classifier and delivers the best one, based on a metric of our choosing. This approach was used on the following classifiers: AdaBoost, Gradient Boosting, MLPClassifier, KNeighborsClassifier, DecisionTreeClassifier, GaussianNB, BaggingClassifier, LogisticRegression and LinearSVC.

The parameters passed as arguments to GridSearch were the scaled datasets (using MinMaxScaler), split by the train-test split method. At that point, AdaBoostClassifier and GradientBoostingClassifier methods scored extremely well and the occurrence of data leakage was noticed and should be addressed.

Data leakage often results in unrealistically-high levels of performance on the test set, because the model is being ran on data that it had already seen — in some capacity — in the training set. The model effectively memorizes the training set data and is easily able to correctly output the labels/values for those test data-set examples. Clearly, this is not ideal, as it misleads the person evaluating the model. (Sony, 2019)

LinearRegression and GaussianNB algorithms, on the other hand, performed quite badly regarding the F1-score, but the latter scored very high at the recall level - the measure that shows how well the classifier can recognize the positive tuples. The recall (or sensitivity) measure is also referred to as the true positive (recognition) rate, i.e., the proportion of positive tuples that are correctly identified (Han, 2012). Thus, both algorithms should not be discarded, due to diversity those methods could introduce in our final prediction outcome.



```
In [64]: # Applying algorithm

         bagging_gnb = BaggingClassifier(base_estimator = model, random_state = 5)

         model = bagging_gnb.fit(scaler_X_train, y_train)

         # Predicting
         y_predicted_train = model.predict(scaler_X_train) # seen data by the model
         y_predicted_val = model.predict(scaler_X_val) # unseen data

         # Checking F1 scores
         print("F1 Score - Train : ", f1_score(y_train, y_predicted_train, average='micro
         print("F1 Score - Test :  ", f1_score(y_val, y_predicted_val, average='micro'))

         F1 Score - Train :  0.4056919642857143
         F1 Score - Test :  0.40133928571428573

In [65]: # Focus at recalls at 1s
         print(classification_report(y_val, y_predicted_val))

                       precision    recall  f1-score   support

                    0       0.96      0.22      0.36      3418
                    1       0.28      0.97      0.44      1062

             accuracy                           0.40      4480
            macro avg       0.62      0.60      0.40      4480
         weighted avg       0.80      0.40      0.38      4480
```

As with many things in Machine Learning, diversity importance mimics the behavior our own brain deals with information. Research by Vinje and Gallan (2000) have shown that the human visual system represents decorrelation and sparseness, namely diversity. This makes different neurons in the human learning respond to different stimuli and generates little redundancy in the learning process which ensures the high effectiveness of the human learning.

In ML, general ensemble learning usually makes the learned multiple base models converge to the same or similar local optima. Thus, diversity among multiple base models is an attempt to repulse different base learners and encourages them to provide choice reflecting multi-modal belief, so the meta-learner can work with multiple diversified choices and significantly improve its performance. (Gong, Zhong, Hu, 2019)

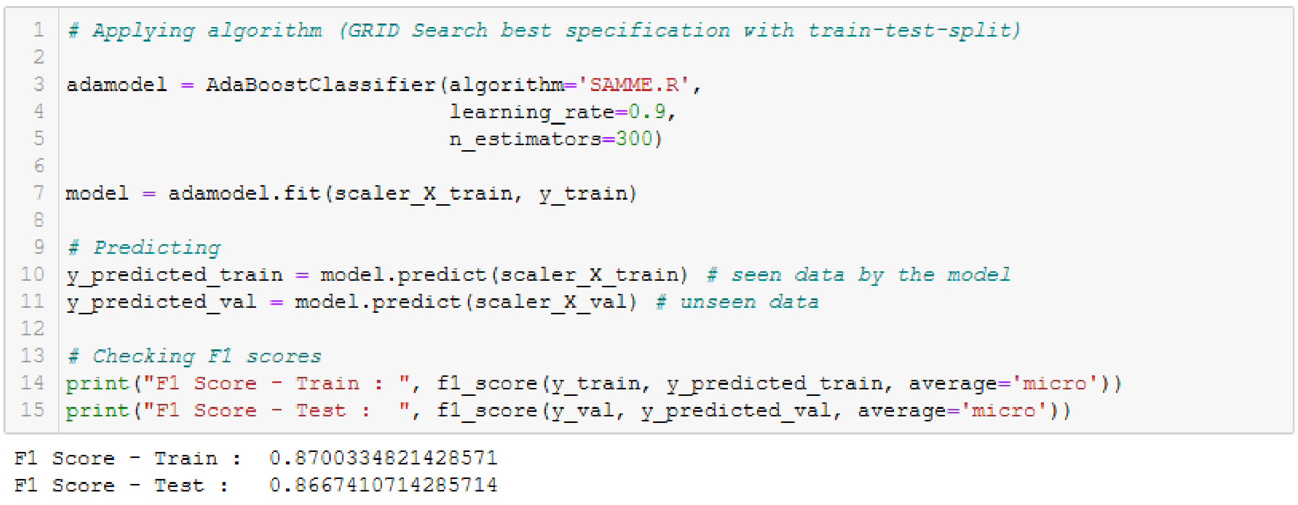
We will cover next which were the algorithms chosen as our Base Learners, and the hyperparameters for each one of them.

Our choice of Base Learners

AdaBoost

Boosting algorithms are an important recent development in classification. These algorithms belong to a group of voting methods that produce a classifier as a linear combination of base or weak classifiers.

Originally, AdaBoost was designed in such a way that at every step the sample distribution was adapted to put more weight on misclassified samples and less weight on correctly classified samples. The final prediction is a weighted average of all the weak learners, where more weight is placed on stronger learners. (Adaboost vs Gradient Boosting, 2018)



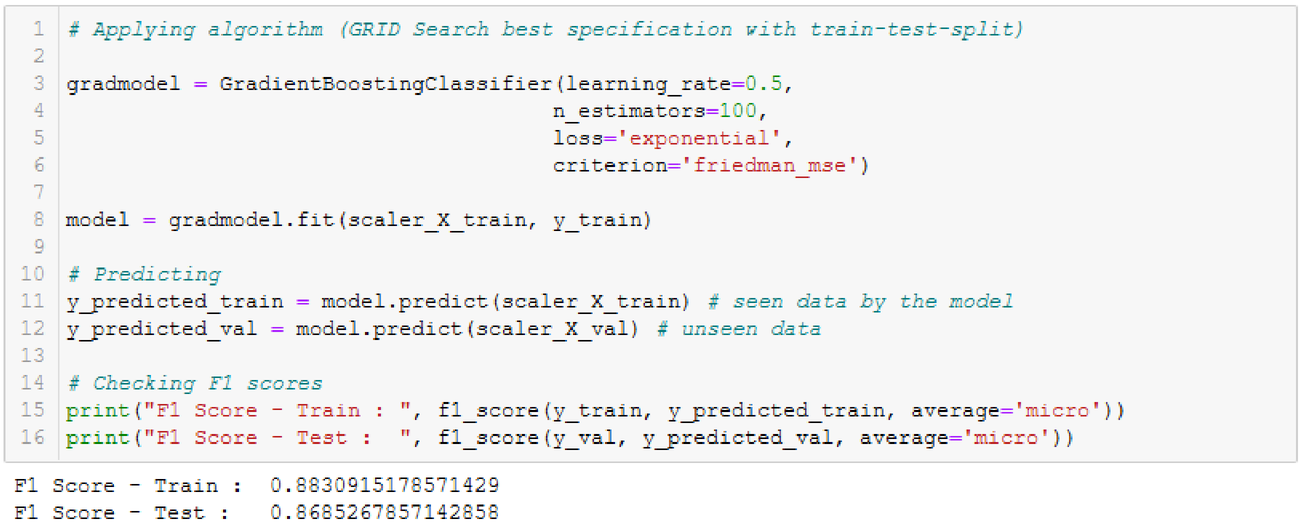
```python
# Applying algorithm (GRID Search best specification with train-test-split)

adamodel = AdaBoostClassifier(algorithm='SAMME.R',
                              learning_rate=0.9,
                              n_estimators=300)

model = adamodel.fit(scaler_X_train, y_train)

# Predicting
y_predicted_train = model.predict(scaler_X_train) # seen data by the model
y_predicted_val = model.predict(scaler_X_val) # unseen data

# Checking F1 scores
print("F1 Score - Train : ", f1_score(y_train, y_predicted_train, average='micro'))
print("F1 Score - Test :  ", f1_score(y_val, y_predicted_val, average='micro'))
```

```
F1 Score - Train :  0.8700334821428571
F1 Score - Test :   0.8667410714285714
```

Gradient Boosting

The term gradient boosting consists of two sub-terms, gradient and boosting. We already know what boosting refers to. Let´s see how the term 'gradient' is related here. Gradient boosting re-defines boosting as a numerical optimization problem where the objective is to minimize the loss function of the model by adding weak learners using gradient descent. Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. As gradient boosting is based on minimizing a loss function, different types of loss functions can be used resulting in a flexible technique that can be applied to regression, multi-class classification, etc. (Mujtaba, 2020)

GRADIENT BOOSTING – Starts by making a leaf with the average, the gradiend boost builds a tree. Like adaboost the tree is basesd on the errors of the previous tree, but the tree is also restricted in size. Gradient Boosting trees are larger than the ones using adaboost: GB builds another tree and scales the tree, and continues to build trees in this fashion until we reach a limit that we may fix or when additional trees fail to improve the fit

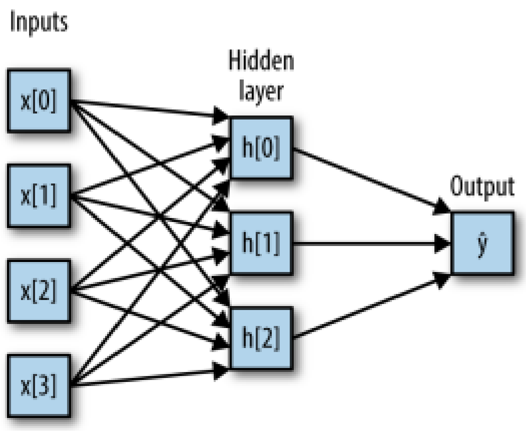


```python
# Applying algorithm (GRID Search best specification with train-test-split)

gradmodel = GradientBoostingClassifier(learning_rate=0.5,
                                       n_estimators=100,
                                       loss='exponential',
                                       criterion='friedman_mse')

model = gradmodel.fit(scaler_X_train, y_train)

# Predicting
y_predicted_train = model.predict(scaler_X_train) # seen data by the model
y_predicted_val = model.predict(scaler_X_val) # unseen data

# Checking F1 scores
print("F1 Score - Train : ", f1_score(y_train, y_predicted_train, average='micro'))
print("F1 Score - Test :  ", f1_score(y_val, y_predicted_val, average='micro'))
```

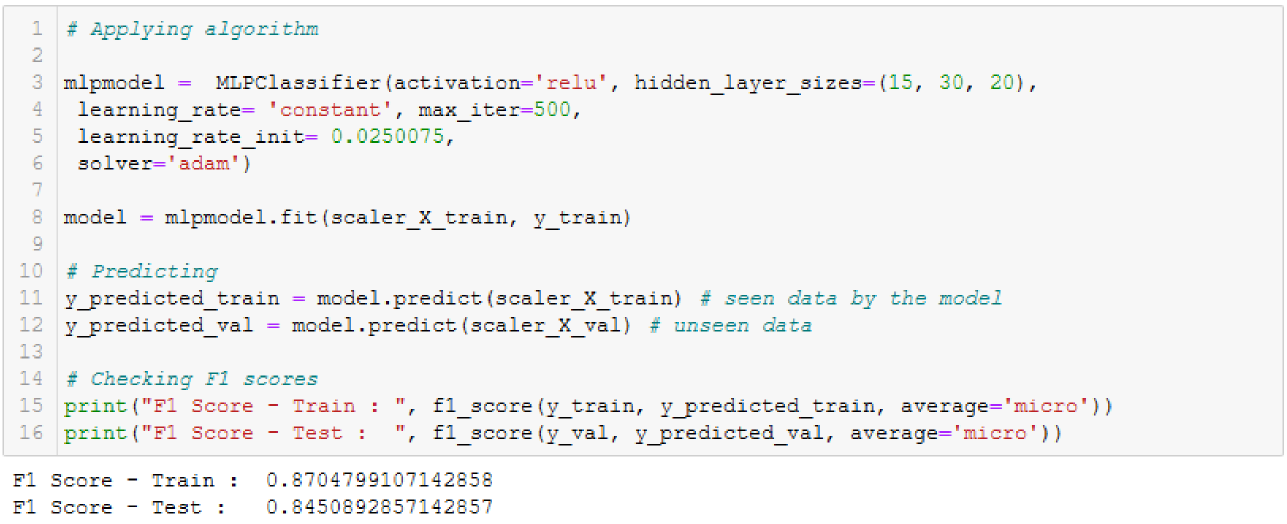
```
F1 Score - Train :  0.8830915178571429
F1 Score - Test :   0.8685267857142858
```

MLPClassifier

MLPs can be viewed as generalizations of linear models that perform multiple stages of processing to come to a decision (Müller, A. & Guido, S., 2017). There are the left nodes, which exemplify the input features, the lines in between represent the learned coefficients, and the right nodes are exemplars of outputs.



There are nodes in between the input and the outputs, which compose the hidden layer. The reasoning here is that the weighted sum of the inputs in the hidden units, we apply a nonlinear function and take the final output.

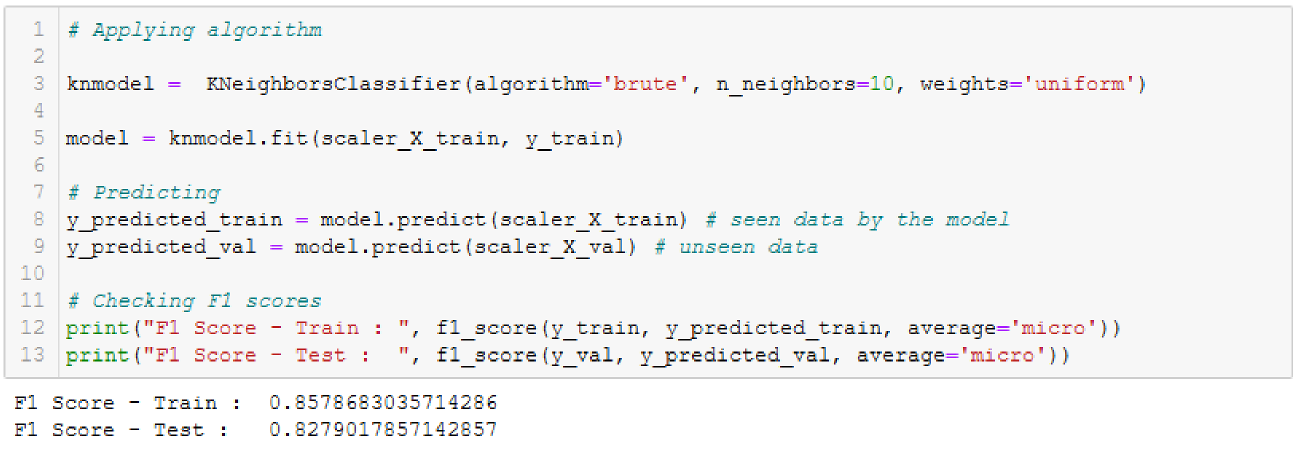


```
# Applying algorithm

mlpmodel =  MLPClassifier(activation='relu', hidden_layer_sizes=(15, 30, 20),
 learning_rate= 'constant', max_iter=500,
 learning_rate_init= 0.0250075,
 solver='adam')

model = mlpmodel.fit(scaler_X_train, y_train)

# Predicting
y_predicted_train = model.predict(scaler_X_train) # seen data by the model
y_predicted_val = model.predict(scaler_X_val) # unseen data

# Checking F1 scores
print("F1 Score - Train : ", f1_score(y_train, y_predicted_train, average='micro'))
print("F1 Score - Test :  ", f1_score(y_val, y_predicted_val, average='micro'))
```

```
F1 Score - Train :  0.8704799107142858
F1 Score - Test :   0.8450892857142857
```

KNeighborsClassifier

K-NN is a classification algorithm and conceptually one of the simplest to understand. It is also called Lazy Learner - as opposed to an Eager Learner. Most classification algorithms are eager learners: there is a set of training data with example classifications. Training data is used to construct classification model. The model is used for evaluation on test data where data classifications are known. If the evaluated results are satisfactory, the final model is then used for making prediction of classes on data with unknown classifications. Eager learners have, therefore, already done most of their job of model formulation beforehand. A lazy learner, on the other hand, does not build any model beforehand; it waits for the unclassified data and then winds its way through the algorithm to make classification prediction. Lazy learners are, therefore, time consuming: each time a prediction must be made all the model building effort must be performed again. (Pandya, 2016)

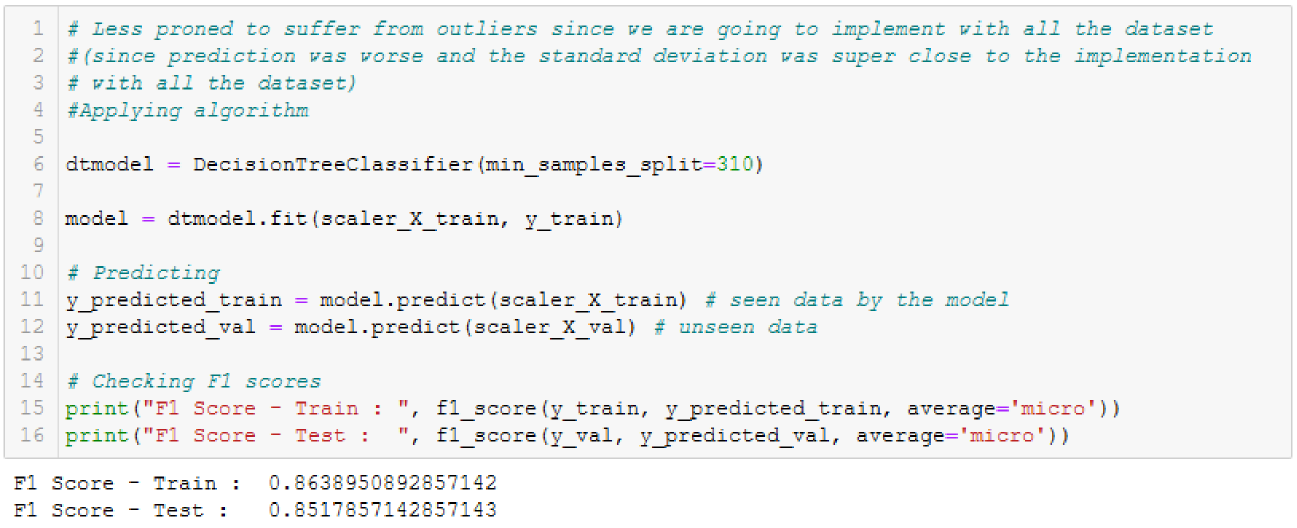


```
# Applying algorithm

knmodel =  KNeighborsClassifier(algorithm='brute', n_neighbors=10, weights='uniform')

model = knmodel.fit(scaler_X_train, y_train)

# Predicting
y_predicted_train = model.predict(scaler_X_train) # seen data by the model
y_predicted_val = model.predict(scaler_X_val) # unseen data

# Checking F1 scores
print("F1 Score - Train : ", f1_score(y_train, y_predicted_train, average='micro'))
print("F1 Score - Test :  ", f1_score(y_val, y_predicted_val, average='micro'))
```

```
F1 Score - Train :   0.8578683035714286
F1 Score - Test :    0.8279017857142857
```

DecisionTreeClassifier

Decision tree is a method which classify labeled trained data into a tree or rules. Once the tree is built then it is pruned to check for overfitting and noise. There are many advantages of using trees: easy to understand and interpret, handles categorical and numeric attributes, are robust to outliers and missing values. (Lavanya & Rani, 2012)

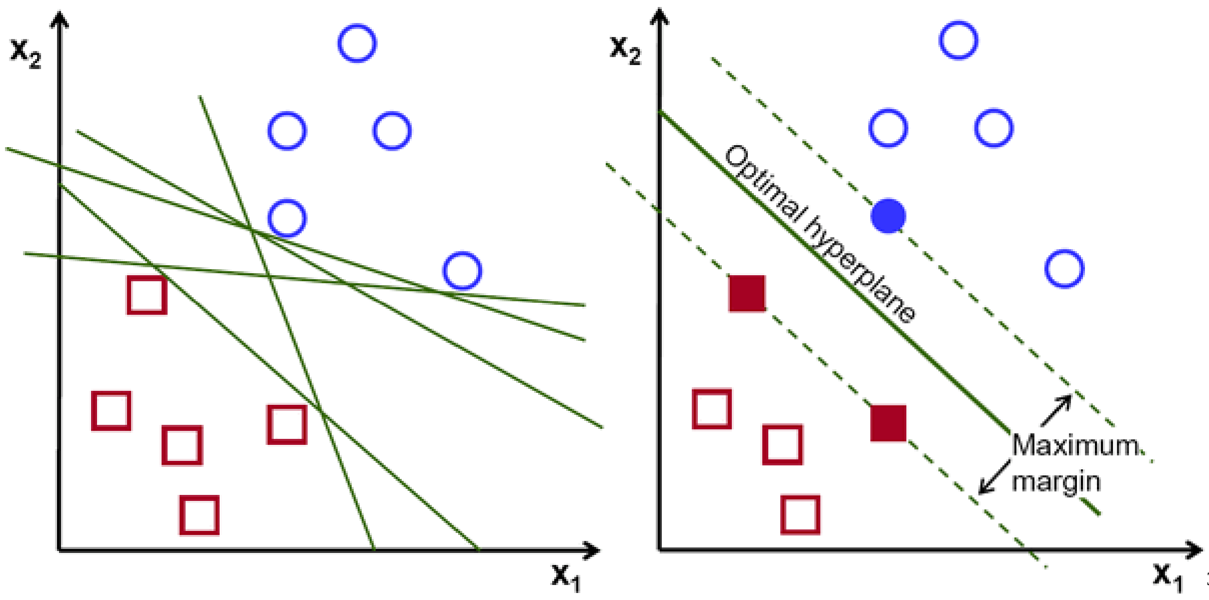


```
# Less proned to suffer from outliers since we are going to implement with all the dataset
#(since prediction was worse and the standard deviation was super close to the implementation
# with all the dataset)
#Applying algorithm

dtmodel = DecisionTreeClassifier(min_samples_split=310)

model = dtmodel.fit(scaler_X_train, y_train)

# Predicting
y_predicted_train = model.predict(scaler_X_train) # seen data by the model
y_predicted_val = model.predict(scaler_X_val) # unseen data

# Checking F1 scores
print("F1 Score - Train : ", f1_score(y_train, y_predicted_train, average='micro'))
print("F1 Score - Test :  ", f1_score(y_val, y_predicted_val, average='micro'))
```

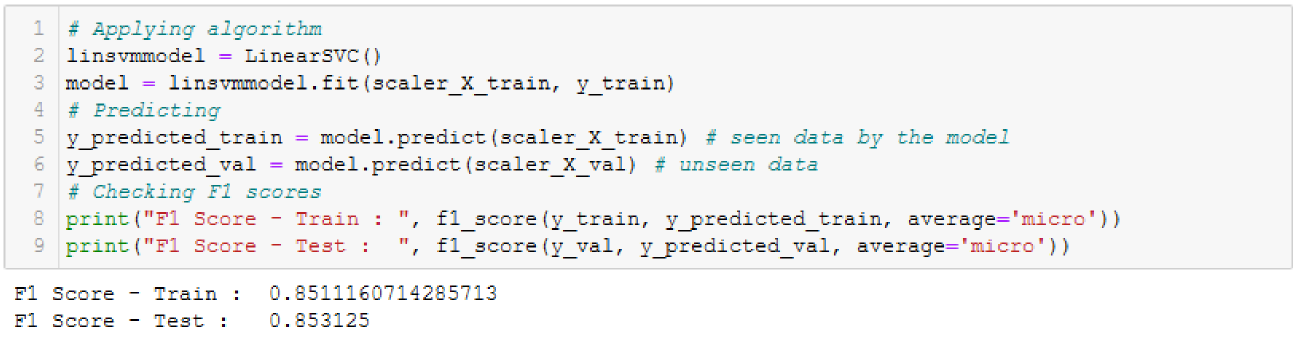
```
F1 Score - Train :   0.8638950892857142
F1 Score - Test :    0.8517857142857143
```

LinearSVC

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space, being N the number of features, that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. The objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. (Gandhi, 2018)



[3]



```
# Applying algorithm
linsvmmodel = LinearSVC()
model = linsvmmodel.fit(scaler_X_train, y_train)
# Predicting
y_predicted_train = model.predict(scaler_X_train) # seen data by the model
y_predicted_val = model.predict(scaler_X_val) # unseen data
# Checking F1 scores
print("F1 Score - Train : ", f1_score(y_train, y_predicted_train, average='micro'))
print("F1 Score - Test :  ", f1_score(y_val, y_predicted_val, average='micro'))
```

```
F1 Score - Train :   0.8511160714285713
F1 Score - Test :    0.853125
```

GaussianNB

Gaussian Naïve Bayes is a set of supervised learning algorithms that apply Bayes' theorem with the "naive" assumption of independence between every pair of features. According to Lou et al. (2014), a NB classifier calculates the probability that a given instance (example) belongs to a certain class. Given an instance X, described by its feature vector and a class target Y, Bayes' theorem allows us to express the conditional probability P(y|X) as a product of simpler probabilities using the naïve independence assumption.

[3] Gandhi, 2018

```python
# Applying algorithm

gnbmodel =  GaussianNB()

model = gnbmodel.fit(scaler_X_train, y_train)

# Predicting
y_predicted_train = model.predict(scaler_X_train) # seen data by the model
y_predicted_val = model.predict(scaler_X_val) # unseen data

# Checking F1 scores
print("F1 Score - Train : ", f1_score(y_train, y_predicted_train, average='micro'))
print("F1 Score - Test :  ", f1_score(y_val, y_predicted_val, average='micro'))
```

```
F1 Score - Train :  0.49430803571428567
F1 Score - Test :   0.49575892857142856
```

```python
# Focus on Recall at 1s
print(classification_report(y_val, y_predicted_val))
```

```
              precision    recall  f1-score   support

           0       0.96      0.35      0.52      3418
           1       0.31      0.96      0.47      1062

    accuracy                           0.50      4480
   macro avg       0.64      0.66      0.49      4480
weighted avg       0.81      0.50      0.51      4480
```

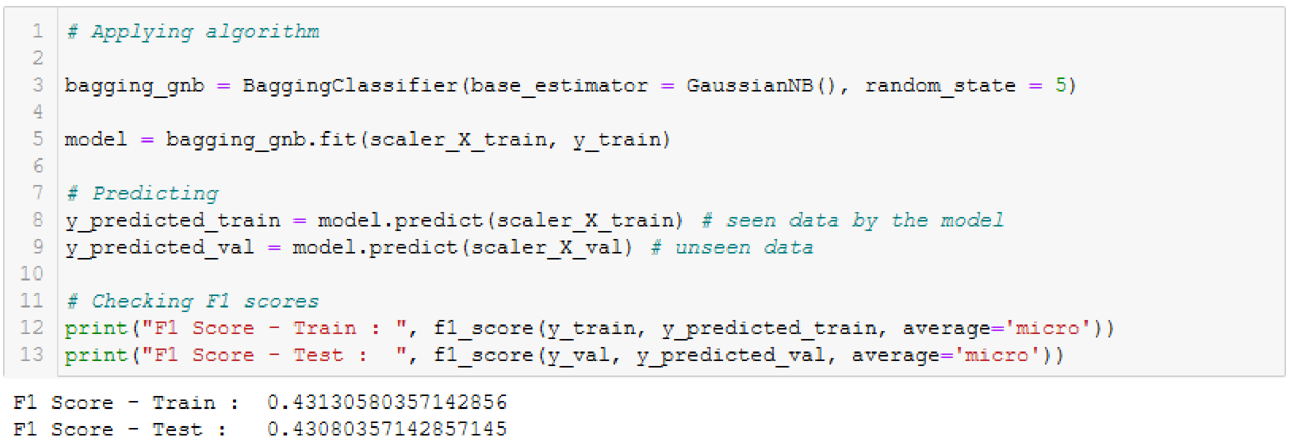
Despite having a low F1 score, GaussianNB had a high value of Recall on 1s, so it is important to bring diversity to our Ensemble Classifier.

BaggingClassifier

Bagging classifier is an ensemble technique which was proposed by Leo Breiman in 1994. It works by combining classifications of randomly generated training sets to form a final prediction. Such techniques can typically be used as a variance reduction technique by randomization into its construction procedure and then creating an ensemble out of it. Bagging classifier has attracted much attention due to its simple implementation and the fact that it improves accuracy. Thus, we can call bagging as a “smoothing operation” that has advantage when intending to improve the predictive performance of regression or classification trees. The basic principle behind of this ensemble method is that a group of “weak learners” can come together to form a “strong learner”.

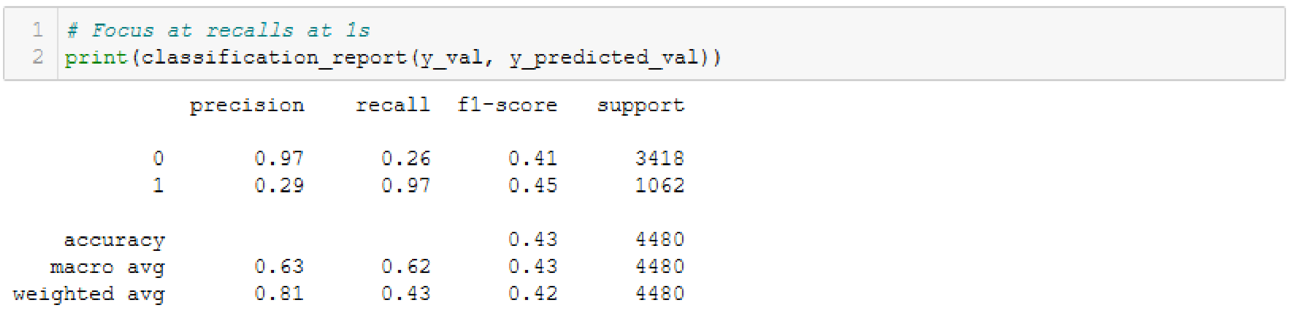
Bagging grows many decision trees. Here each individual decision tree is a “weak learner”, while all the decision trees taken together are a “strong learner”. When a new instance has to be classified, it is done repeatedly to each of the trees in the ensemble. Each tree gives a “vote” for a class. The final prediction for the new instance’s class is gained by the class having maximum votes. (Zareapoor & Shamsolmoali, 2015)

In the example below, we have a small increment on the capacity to identify recall of 1s and precision of 0s, but the overall F1 score has decreased, in comparison to just the GaussianNB analysis previously done.



```python
# Applying algorithm

bagging_gnb = BaggingClassifier(base_estimator = GaussianNB(), random_state = 5)

model = bagging_gnb.fit(scaler_X_train, y_train)

# Predicting
y_predicted_train = model.predict(scaler_X_train) # seen data by the model
y_predicted_val = model.predict(scaler_X_val) # unseen data

# Checking F1 scores
print("F1 Score - Train : ", f1_score(y_train, y_predicted_train, average='micro'))
print("F1 Score - Test :  ", f1_score(y_val, y_predicted_val, average='micro'))
```

```
F1 Score - Train :  0.43130580357142856
F1 Score - Test :   0.43080357142857145
```



```python
# Focus at recalls at 1s
print(classification_report(y_val, y_predicted_val))
```

```
              precision    recall  f1-score   support

           0       0.97      0.26      0.41      3418
           1       0.29      0.97      0.45      1062

    accuracy                           0.43      4480
   macro avg       0.63      0.62      0.43      4480
weighted avg       0.81      0.43      0.42      4480
```

LogisticRegression

Logistic Regression is one of the most simple and commonly used Machine Learning algorithms for two-class classification. It is easy to implement and can be used as the baseline for any binary classification problem. Its basic fundamental concepts are also constructive in deep learning. Logistic regression describes and estimates the relationship between one dependent binary variable and independent variables, by computing the probability of an event occurring. (Navlani, 2019)

```
# Applying algorithm
lrmodel = LogisticRegression(penalty='elasticnet', solver='saga', l1_ratio=0.9)
model = lrmodel.fit(scaler_X_train, y_train)
# Predicting
y_predicted_train = model.predict(scaler_X_train) # seen data by the model
y_predicted_val = model.predict(scaler_X_val) # unseen data
# Checking F1 scores
print("F1 Score - Train : ", f1_score(y_train, y_predicted_train, average='micro'))
print("F1 Score - Test :  ", f1_score(y_val, y_predicted_val, average='micro'))
```

```
F1 Score - Train :  0.8514508928571429
F1 Score - Test :   0.8544642857142857
```

Not used because did not bring diversity to the stacking implementation

```
print(classification_report(y_val, y_predicted_val))
```

```
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      3418
           1       0.74      0.59      0.66      1062

    accuracy                           0.85      4480
   macro avg       0.81      0.76      0.78      4480
weighted avg       0.85      0.85      0.85      4480
```

Meta Learner

Now that we have tested and selected the best algorithms with their hyperparameters set, we will use a meta learner to further improve our score by combining different models into one.

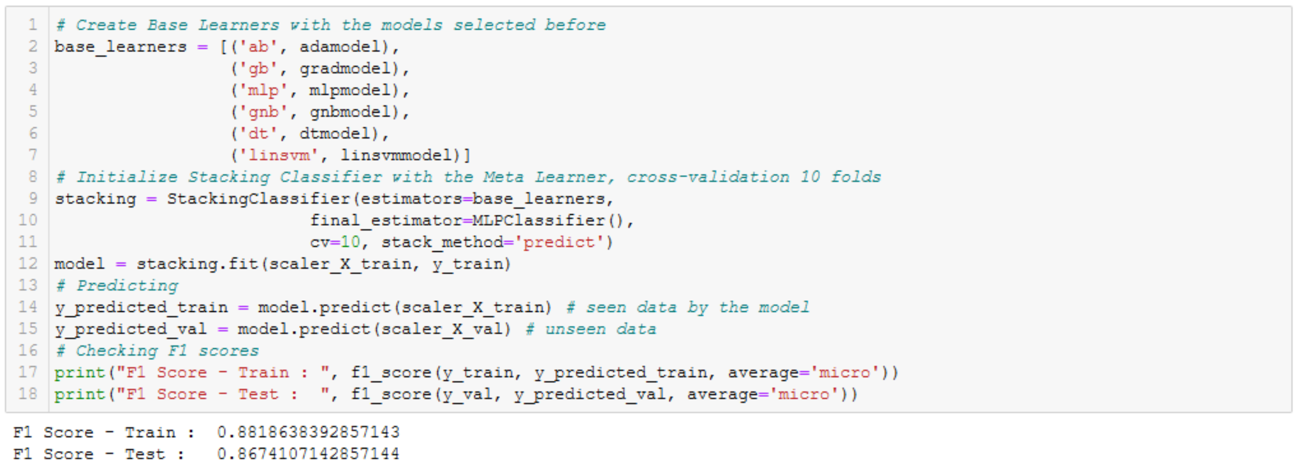
Ensemble methods - improving classification accuracy:

Ensemble methods is a technique which tends to increase predictions accuracy by joining the results from different classifiers. The motivation to adopt them is due to fact that different classifiers can yield different predictions regarding their specific characteristics and different sensitivity to random values on training data when applied individually. Usually, ensembles yield better results when there is significant diversity among the models, .i.e., there is little correlation among classifiers. (Hasen & Salamon, 1990)

The logic in ensemble analysis can be better understood by examining the different components of the error of the classifier. According to Aggarwal (2015), there are three primary components to a classifier error: i) bias - every classifier makes its own modeling assumptions about the nature of the decision boundary between classes; ii) Variance - random variations in the choices of the training data will lead to different models, and iii) Noise, which refers to the intrinsic errors in the target class labeling. So, it leads to a conclusion that the design choices of a classifier often reflect a trade-off between the bias and the variance.

The basic process of ensemble analysis is to apply the base ensemble learners multiple times by using different models, or by using the same model on different subsets of the training data. Thus, the results from different classifiers are then combined into a single solid prediction.

Different combination of the base learners covered before were tested, and in the end the best results came from this mix: AdaBoost, MLPClassifier, GaussianNB, DecisionTreeClassifier, LinearSVC.



```
# Create Base Learners with the models selected before
base_learners = [('ab', adamodel),
                 ('gb', gradmodel),
                 ('mlp', mlpmodel),
                 ('gnb', gnbmodel),
                 ('dt', dtmodel),
                 ('linsvm', linsvmmodel)]
# Initialize Stacking Classifier with the Meta Learner, cross-validation 10 folds
stacking = StackingClassifier(estimators=base_learners,
                              final_estimator=MLPClassifier(),
                              cv=10, stack_method='predict')
model = stacking.fit(scaler_X_train, y_train)
# Predicting
y_predicted_train = model.predict(scaler_X_train) # seen data by the model
y_predicted_val = model.predict(scaler_X_val) # unseen data
# Checking F1 scores
print("F1 Score - Train : ", f1_score(y_train, y_predicted_train, average='micro'))
print("F1 Score - Test :  ", f1_score(y_val, y_predicted_val, average='micro'))
```

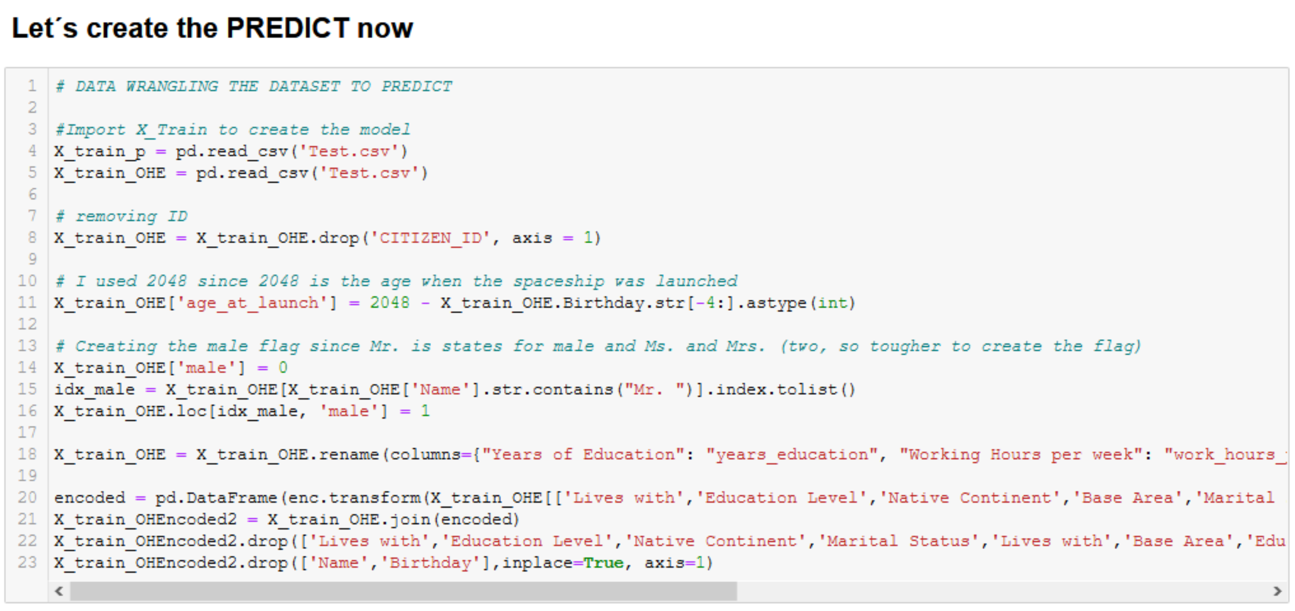
```
F1 Score - Train :  0.8818638392857143
F1 Score - Test :   0.8674107142857144
```

VI. Deployment - Predicting Newland

With the best model trained (*fit*), the next step is to load the "Test.csv".

We performed the same data wrangling done before with *Train* dataset: Feature Engineering, OHE, Scaling.

Next we *transform* the *Test* dataset using the model fitted above.



Let´s create the PREDICT now

```
# DATA WRANGLING THE DATASET TO PREDICT

#Import X_Train to create the model
X_train_p = pd.read_csv('Test.csv')
X_train_OHE = pd.read_csv('Test.csv')

# removing ID
X_train_OHE = X_train_OHE.drop('CITIZEN_ID', axis = 1)

# I used 2048 since 2048 is the age when the spaceship was launched
X_train_OHE['age_at_launch'] = 2048 - X_train_OHE.Birthday.str[-4:].astype(int)

# Creating the male flag since Mr. is states for male and Ms. and Mrs. (two, so tougher to create the flag)
X_train_OHE['male'] = 0
idx_male = X_train_OHE[X_train_OHE['Name'].str.contains("Mr. ")].index.tolist()
X_train_OHE.loc[idx_male, 'male'] = 1

X_train_OHE = X_train_OHE.rename(columns={"Years of Education": "years_education", "Working Hours per week": "work_hours_

encoded = pd.DataFrame(enc.transform(X_train_OHE[['Lives with','Education Level','Native Continent','Base Area','Marital
X_train_OHEncoded2 = X_train_OHE.join(encoded)
X_train_OHEncoded2.drop(['Lives with','Education Level','Native Continent','Marital Status','Lives with','Base Area','Edu
X_train_OHEncoded2.drop(['Name','Birthday'],inplace=True, axis=1)
```

```python
# Fit the scaler to TEST data, then transform
#scaler_ = MinMaxScaler().fit(X_train_OHEncoded2)
# I think it makes more sense to use the fit out the trained model, not with all the test dataset that we are going to pr
scaler_X_train_ = scaler.transform(X_train_OHEncoded2)

# Predicting
submission = model.predict(scaler_X_train_)
X_train['Income'] = submission
SUBMIT = X_train[['CITIZEN_ID','Income']]
SUBMIT.head()
```

	CITIZEN_ID	Income
0	34886	1
1	34887	0
2	34888	1
3	34889	0
4	34890	0

```python
# Export to CSV to update to Kaggle
SUBMIT.to_csv(path_or_buf="Export.csv", index=False)
SUBMIT.shape # Check the shape of exported file. It must be (10100, 2).
```

```
(10100, 2)
```

That output is the one we submitted on Kaggle. The results will be out soon!

REFERENCES

Adaboost vs Gradient Boosting. (2018, October 4). Data Science Stack Exchange. https://datascience.stackexchange.com/questions/39193/adaboost-vs-gradient-boosting

Aggarwal, C. C. (2015). *Data mining: the textbook*. Springer.

Berry, M. J., & Linoff, G. S. (2004). Data mining techniques: for marketing, sales, and customer relationship management. John Wiley & Sons.

Cerda, P, Varoquaux, G. & Kégl, B. (2018). Similarity encoding for learning with dirty categorical variables. Mach Learn 107, 1477–1494

Corder, G. W., & Foreman, D. I. (2011). Nonparametric statistics for non-statisticians.

D'Agostino, R. B. (1986). Tests for the normal distribution. Goodness-of-fit techniques, 367-419.

Gandhi, R. (2018, July 5). Support Vector Machine — Introduction to Machine Learning Algorithms. Towards Data Science. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Ghosh, A., Manwani, N., & Sastry, P. S. (2017, May). On the robustness of decision tree learning under label noise. In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 685-697). Springer, Cham.

Gong, Z., Zhong, P., & Hu, W. (2019). Diversity in machine learning. IEEE Access, 7, 64323-64350.

Han, Jiawei & Kamber, Micheline & Pei, Jian. (2012). Data Mining: Concepts and Techniques. 10.1016/C2009-0-61819-5.

Hasen, L., & Salamon, P. (1990). Neural networks ensembles. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, *12*(10), 993-1001.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.

Kelleher, J. D., Mac Namee, B., & D'arcy, A. (2015). Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies. MIT press.

Lamrini, B., Gjini, A., Daudin, S., Pratmarty, P., Armando, F., & Travé-Massuyès, L. (2018). Anomaly Detection using Similarity-based One-Class SVM for Network Traffic Characterization. In DX@ Safeprocess.

Lavanya, D., & Rani, K. U. (2012). Ensemble decision tree classifier for breast cancer data. International Journal of Information Technology Convergence and Services, 2(1), 17.

Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008, December). Isolation forest. In 2008 Eighth IEEE International Conference on Data Mining (pp. 413-422). IEEE.

Lou, W., Wang, X., Chen, F., Chen, Y., Jiang, B., & Zhang, H. (2014). Sequence based prediction of DNA-binding proteins based on hybrid feature selection using random forest and Gaussian naive Bayes. PloS one, 9(1), e86703.

Mujtaba, H. (2020, June 8). What is Gradient Boosting and how is it different from AdaBoost? GreatLearning. https://www.mygreatlearning.com/blog/gradient-boosting/

Müller, A. & Guido, S. (2017). Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media.

Navlani, A. (2019, December 16). Understanding Logistic Regression in Python. Datacamp. https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

Ng, P. (2017). dna2vec: Consistent vector representations of variable-length k-mers. arXiv preprint arXiv:1701.06279.
Outlier detection with Local Outlier Factor (LOF) — scikit-learn 0.24.0 documentation. (2020). Scikit Learn. https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html

Pandya, V. J. (2016, December). Comparing Handwritten Character Recognition by AdaBoostClassifier and KNeighborsClassifier. In 2016 8th International Conference on Computational Intelligence and Communication Networks (CICN) (pp. 271-274). IEEE.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. Neural computation, 13(7), 1443-1471.

Soni, D. (2019, July 16). Data Leakage in Machine Learning - Towards Data Science. Towards Data Science. https://towardsdatascience.com/data-leakage-in-machine-learning-10bdd3eec742

W. E. Vinje and J. L. Gallant (2000). "Sparse coding and decorrelation in primary visual cortex during natural vision", Science, vol. 287, no. 5456, pp. 1273-1276.

Wood, T. (2020, August 7). F-Score. DeepAI. https://deepai.org/machine-learning-glossary-and-terms/f-score

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., et al. (2008). Top 10 algorithms in data mining. Knowledge and information systems, 14(1), 1-37.

Zareapoor, M., & Shamsolmoali, P. (2015). Application of credit card fraud detection: Based on bagging ensemble classifier. Procedia computer science, 48(2015), 679-685.

Zhang, Z., Mayer, G., Dauvilliers, Y. Plazzi, G., Pizza, F., et al. (2018), A simple example of visualizing gradient boosting, Exploring the clinical features of narcolepsy type 1 versus narcolepsy type 2 from European Narcolepsy Network database with machine learning.