Machine Learning final Assignment

Lorenzo Bozzoni

May 2022

Contents

More detailed information regarding programming choices is present in the Python code

1 Introduction

Banks earn a major revenue from lending loans. But it is often associated with risk. The borrower's may default on the loan. To mitigate this issue, the banks have decided to use Machine Learning to overcome this issue. They have collected past data on the loan borrowers & would like you to develop a strong ML Model to classify if any new borrower is likely to default or not.

2 Operative steps

2.1 Data understanding and visualization

The first step that I found necessary and useful to understand the data set was to obtain information on the size, type of characteristics and their distribution, also through the visualization of the data. To do this I used a pandas method which produced the following output:

```
Data columns (total 32 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   loan_limit                 145326 non-null  object
 1   Gender                     148670 non-null  object
 2   approv_in_adv              147762 non-null  object
 3   loan_type                  148670 non-null  object
 4   loan_purpose               148536 non-null  object
 5   Credit_Worthiness          148670 non-null  object
 6   open_credit                148670 non-null  object
 7   business_or_commercial     148670 non-null  object
 8   loan_amount                148670 non-null  int64
 9   rate_of_interest           112231 non-null  float64
 10  Interest_rate_spread       112031 non-null  float64
 11  Upfront_charges            109028 non-null  float64
 12  term                       148629 non-null  float64
 13  Neg_ammortization          148549 non-null  object
 14  interest_only              148670 non-null  object
 15  lump_sum_payment           148670 non-null  object
 16  property_value             133572 non-null  float64
 17  construction_type          148670 non-null  object
 18  occupancy_type             148670 non-null  object
 19  Secured_by                 148670 non-null  object
 20  total_units                148670 non-null  object
 21  income                     139520 non-null  float64
 22  credit_type                148670 non-null  object
 23  Credit_Score               148670 non-null  int64
 24  co-applicant_credit_type   148670 non-null  object
 25  age                        148470 non-null  object
 26  submission_of_application  148470 non-null  object
 27  LTV                        133572 non-null  float64
 28  Region                     148670 non-null  object
 29  Security_Type              148670 non-null  object
 30  Status                     148670 non-null  int64
 31  dtir1                      124549 non-null  float64
dtypes: float64(8), int64(3), object(21)
```

Listing 1: List of features

Subsequently, I represented the categorical features through bar diagrams and the numerical features with histograms.

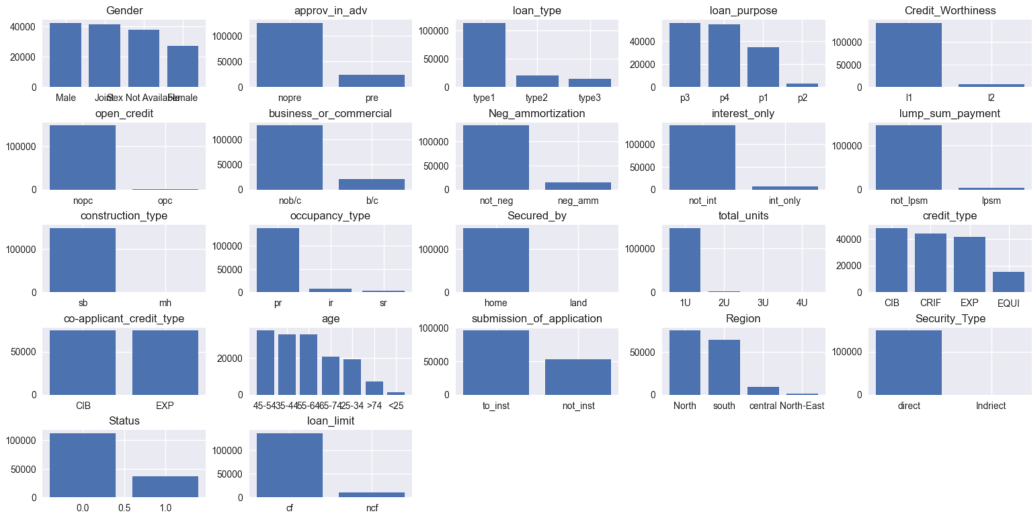

Figure 1: Categorical features

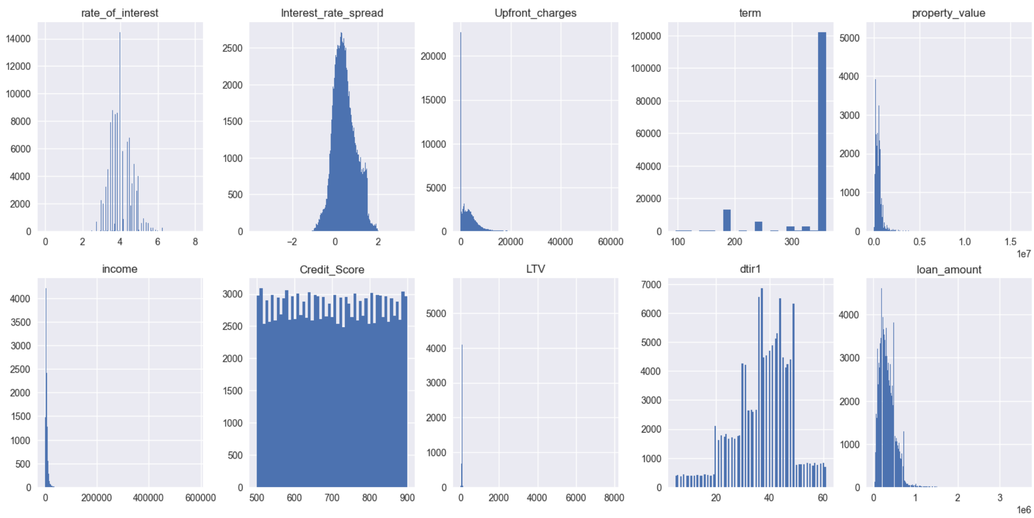

Figure 2: Numerical features

2.2 Data manipulation

Before you can leverage machine learning models and have predictive results, you need to prepare your data for that purpose. This operation can be divided into several phases:

- conversion of categorical features into numerical ones
- identification and removal of outliers
- removal of null values
- feature selection
- feature scaling
- dimensionality reduction

2.2.1 Conversion of categorical features into numerical

```python
# substituting categorical values with numerical ones
for (columnName, columnData) in df.iteritems():
    if columnData.nunique() <= 7 and columnName != "Status":
        freqSeries = columnData.value_counts()
```

```
        dfNumeric[columnName].replace(columnData.value_counts().index,np.arange(0, columnData.
    nunique()),inplace=True)
```

Listing 2: Convert categorical values to numerical

2.2.2 Detection and removal of outliers

In this phase, the values that differ greatly from the statistical average, which are not therefore representative of the sample considered, have been eliminated. It is important to do this as the values that are removed often cause the models to overfit. Specifically, I analyzed the numerical features and, as a discriminant for the selection of values, I used the percentile.

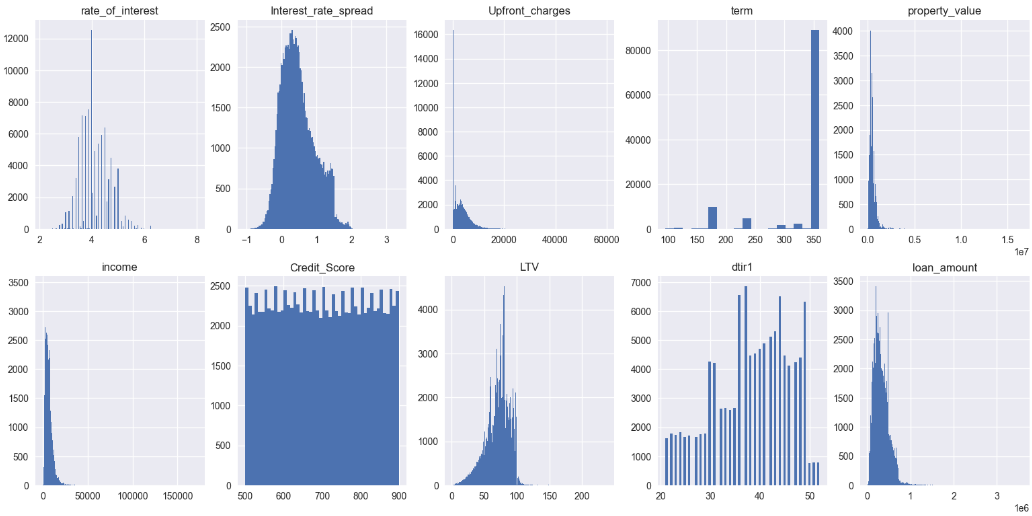

Figure 3: Numerical features without outliers

It is possible to notice, by viewing the histograms, how the values of the abscissas are in smaller ranges, thus making the graph more consistent. Take, for example, the "LTV" feature.

```
# removing outvalues
for (columnName, columnData) in df.iteritems():
    if columnData.nunique() > 7 and columnName != "Status":
        max_thresold = dfNumeric[columnName].quantile(0.995)
        min_thresold = dfNumeric[columnName].quantile(0.005)
        dfNumeric = df[(df[columnName] < max_thresold) & (df[columnName] > min_thresold)]
```

Listing 3: Code for removing outliers

2.2.3 Removal of null values

This was done using the methods offered by Pandas. In the first place I tried to understand what was the distribution of null values, also by means of the graphics library.

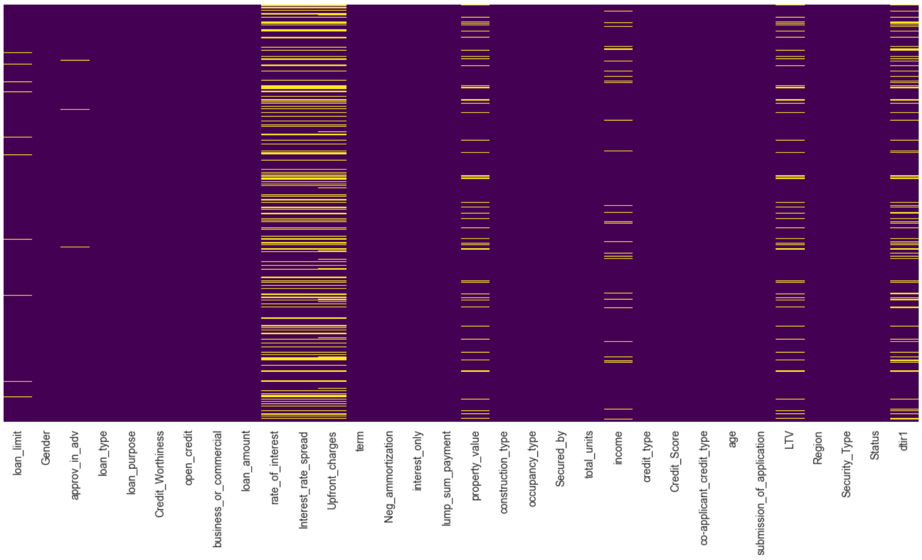

Figure 4: Distribution of null values

In fact, it can be seen that there are mainly the 3 columns ["rate_of_interest","interest_rate_spread","upfront_charges"] having a considerable percentage of null values I therefore decided to visualize the grid representing the Pearson correlation between features as, if it proves strong, it could be exploited to deduce the missing values.

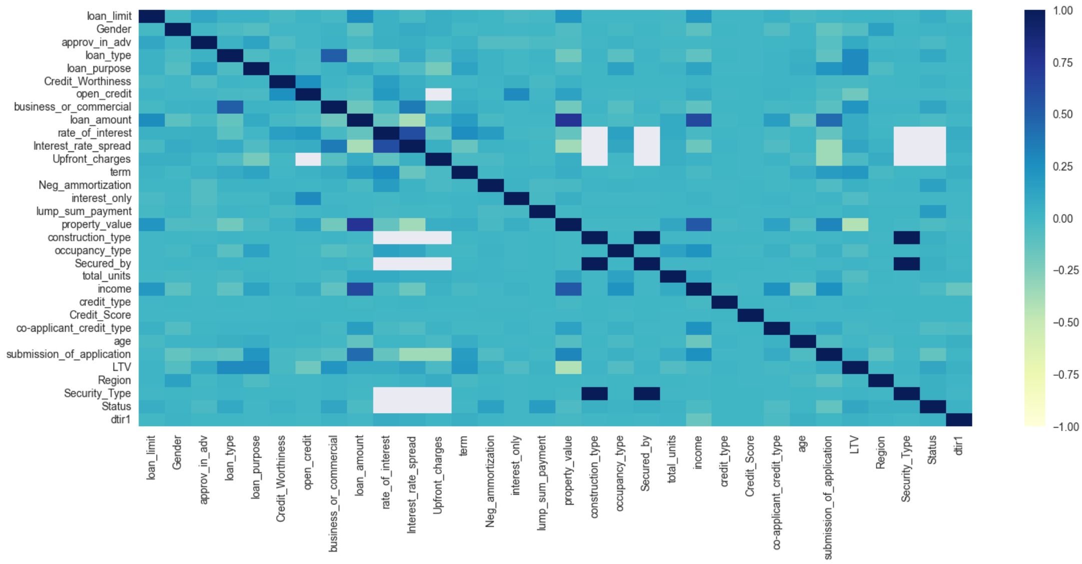

Figure 5: Features correlation

As you can see, the results obtained from the correlation operation are very "low" or rather far from the extremes + or - 1. For this reason, combined with the fact that I am not an expert in the application domain, I have decided that the substitution of the null values with the average calculated for each features, could be, at least in the first place, an acceptable solution.

```
# replacing null values with the mean of the respective feature
dfNumeric.fillna(dfNumeric.mean(),inplace=True)
```

Listing 4: Substituing null values with mean of the feature

2.2.4 Feature selection

Feature selection is a supervised task that consists of removing features that are deemed not to make a minimal useful contribution to target calculation. When you have in-depth knowledge of the reference field of the dataset it is easier to do this. However, as this was not the case, I had to rely on the information extracted from the data itself. For example, I immediately evaluated that the ”ID” and ”Year” features could be eliminated. Later I noticed the presence of other features of a single value ([”construction_type”, ”Secured_by”, ”Security_Type”]) which I proceeded to remove as the statistical contribution of these, it was null.

```
df.drop(["ID", "year"], axis=1, inplace=True)
dfNumeric = dfNumeric.drop(["construction_type","Secured_by","Security_Type"], axis=1)
```

Listing 5: Removing features with no statistical contribution

2.2.5 Dimensionality reduction e feature scaling

Once all the data cleaning activities were carried out, since the dataset used was large, I took advantage of the method offered by sklearn to apply the PCA (I did not use chi2 because of negative values). Before doing this, it is necessary to resize the values assumed by the features themselves, since, representing different variables, the ranges of values vary substantially. The standardization operation (or Z-score) solves that problem by making all the values with zero mean and standard deviation unitary with the following calculation:

$$\textbf{Standardization (or Z-Score)} : x' = \frac{x - \overline{x}}{\sigma} \tag{1}$$

By selecting the percentage value of information to be kept around 95 %, the algorithm has produced a dataframe having 28 features, instead of the starting 31. Reducing the number of columns helps to avoid Curse of Dimensionality and facilitate gradient descent.

2.3 Modelling part

Now the data is clean, you can proceed to the modeling part. I decided to proceed by dividing the training set data from the test set data first. From the first of these, I took advantage of the gridsearch because, although less efficient than the random search, it allows you to view the results obtained by varying the hyperparameters in a very understandable and intuitive way. As it is possible to see later in fact, almost for all the models, I have plotted the trend of the accuracy with the variation of two parameters obtained also using the kfold validation. The values that maximize the precision of the estimates are saved and used in the second phase, i.e. the one in which each model is evaluated with the previously stored test set.

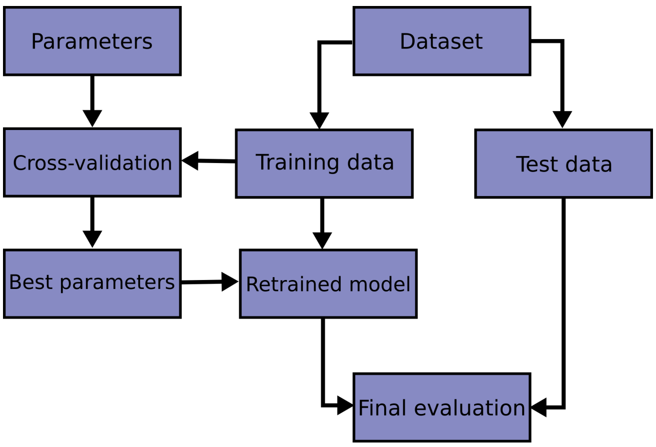

Figure 6: Grid search workflow

The main problem with this dataset is that it is highly unbalanced in fact there are many more labels of one class than the other. To overcome this limit I decided to use SMOTE (Synthetic Minority Over-sampling Technique), which is a technique that allows you to create new samples for the minority class. In this way we obtained a balanced starting dataset and, later, when it was necessary to further subdivide the data, we took advantage of the ”stratify” property whose purpose is to maintain the proportion of the classes.

This allowed me to get balanced train sets, test sets and subsets of the kfold. Having this quality, the accuracy metric value becomes more consistent and can therefore be used to compare different models and parameters.

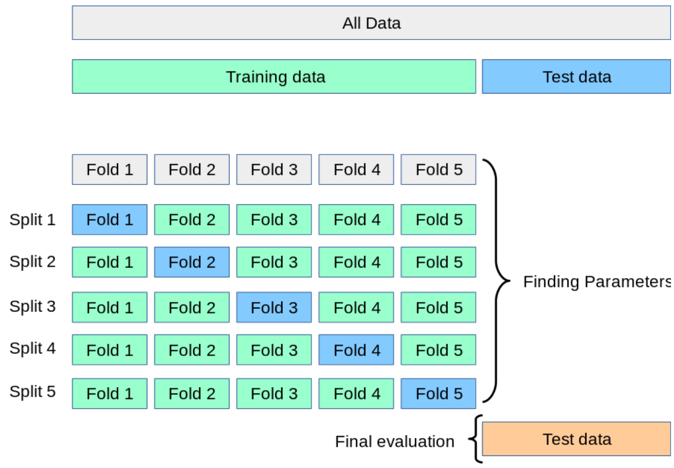

Figure 7: Grid search cross validation

2.3.1 Logistic Regression

In this case, since there are no parameters, the kfold cross validation was simply applied to obtain an average result of the model precision.

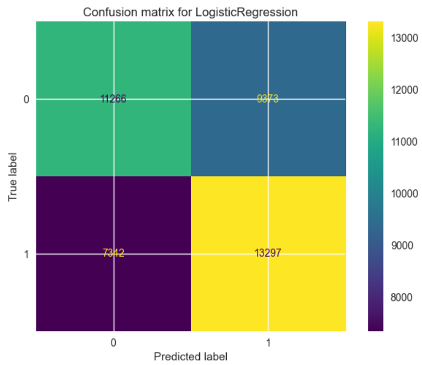

Figure 8: Logistic Regression Confusion Matrix

2.3.2 Decision Tree Classifier

For this model, the parameters that can be varied are many. I have decided to mnanipulate min_sample_leaf and max_depth: the first represents the minimum number of elements that must be present in a decision tree leaf, while the second indicates its maximum depth. Both values are also used to limit overfitting.

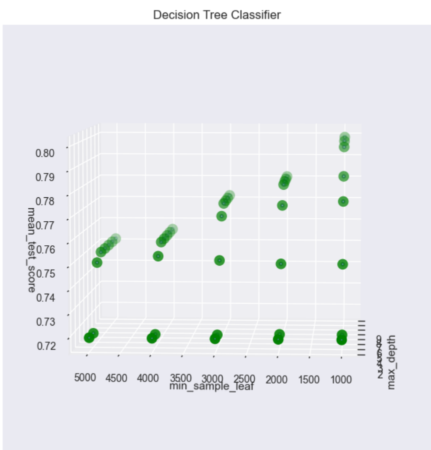

Figure 9: Decision Tree Classifier

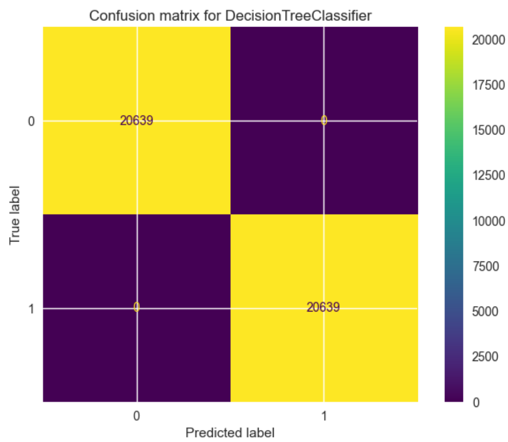

Figure 10: Decision Tree Confusion Matrix

2.3.3 AdaBoost Classifier

The idea behind boosting is to train the predictors sequentially. Each predictor tries a correct its predecessor. One way to correct the predecessor, the one exploited by the AdaBoost model, is to pay attention to the training instances that have been "underfitted" previously.

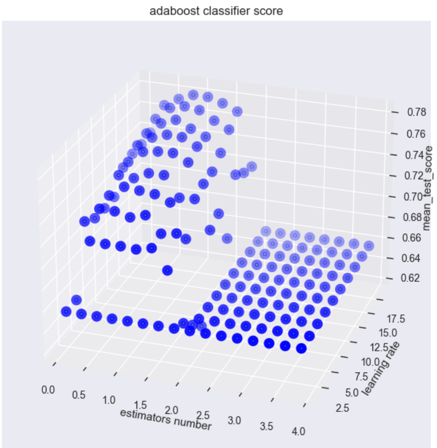

Figure 11: AdaBoost Classifier

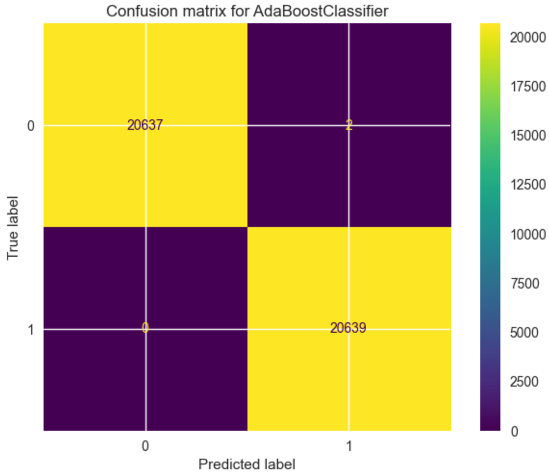

Figure 12: AdaBoost Confusion Matrix

2.3.4 Gradient Boosting Classifier

Boosting consists in training the predictors sequentially so the hyperparameters can define, for example, the number of models to use. Furthermore, it is possible to vary the learning_rate, that is the ratio between speed and precision that you want to have in training the model. A low rate will allow you to find the minimum point in the gradient descent more precisely but computationally it will be much heavier.

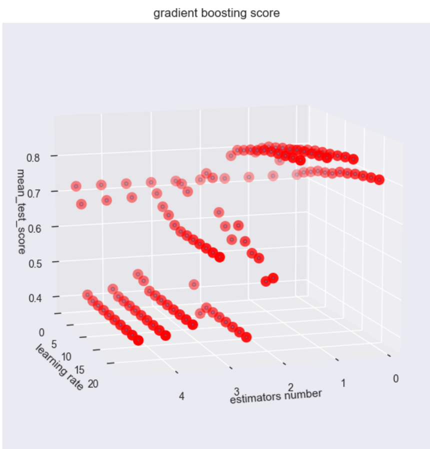

Figure 13: Gradient Boosting Classifier

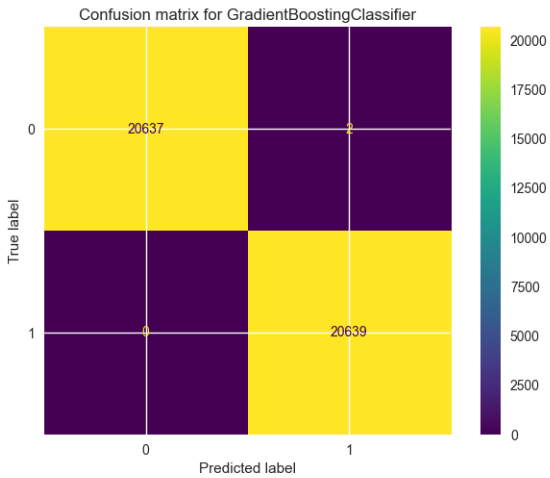

Figure 14: Gradient Boosting Confusion Matrix

2.3.5 Bagging Classifier

Unlike boosting, bagging involves using several predictors in parallel. In this case Logistic Regression are used. The Boolean value associated with the Bootstrap property has been encoded in the graph to values 0 for False and 1 for True.

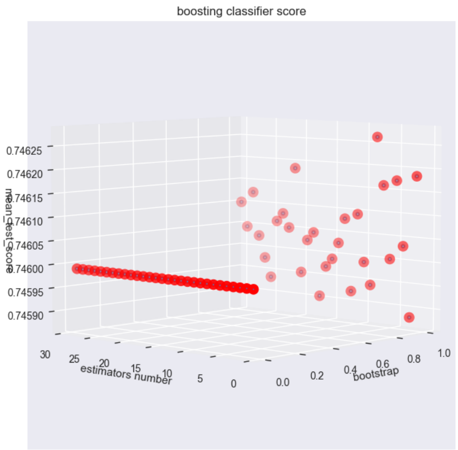

Figure 15: Bagging Classifier

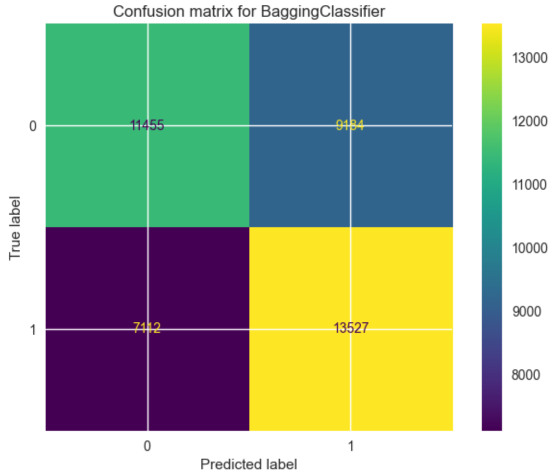

Figure 16: Bagging Confusion Matrix

2.3.6 Neural Network

Considering the not small size of the dataset, I decided to use a neural network. The NN used is the default one, it has a single hidden layer therefore, adding the initial and final one, there are 3 layers in total. the best. You can see from the graphs how by increasing the number of epochs or the number of complete passes through the train set, the result improves considerably.

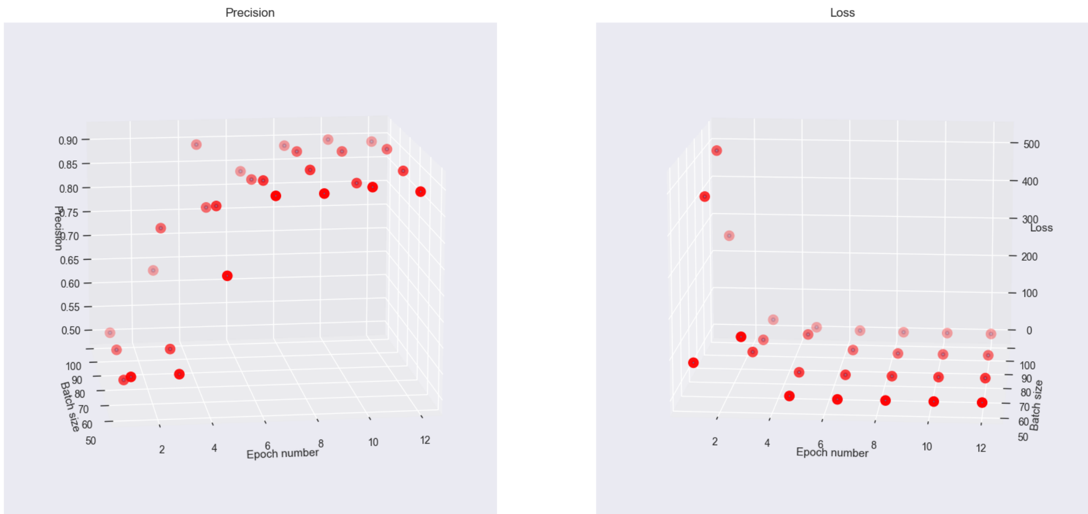

Figure 17: Neural Network metrics

2.4 Model Comparison

The results obtained are the following:

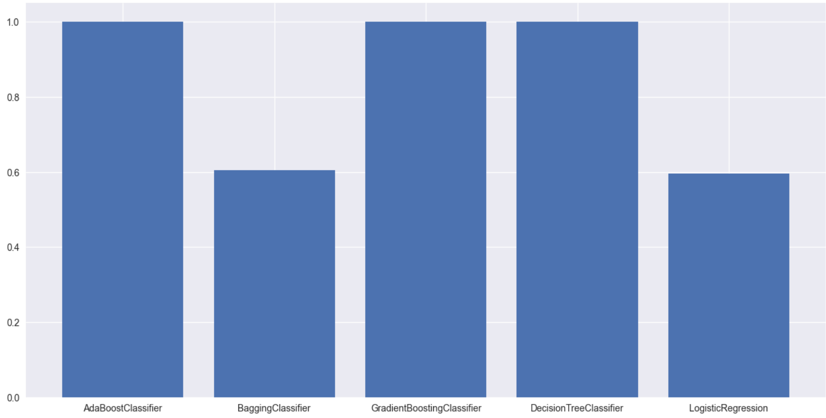

Figure 18: Accuracy score

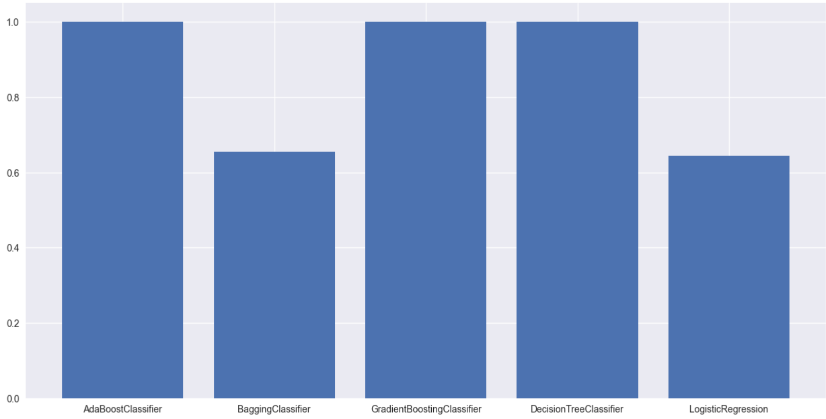

Figure 19: Recall score

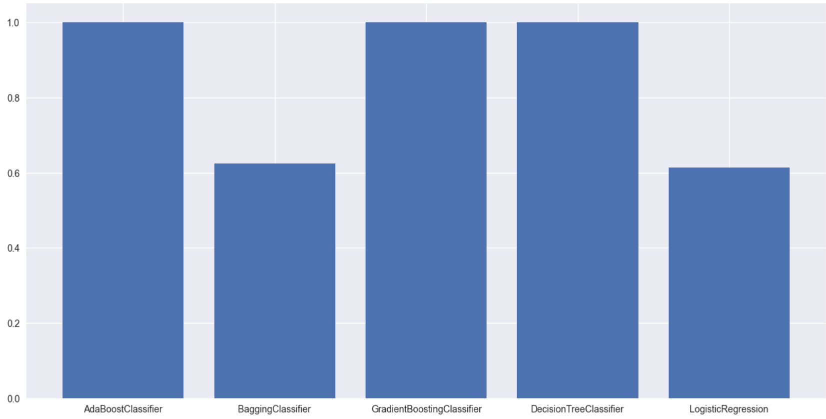

Figure 20: F1 score

3 Conclusions

As can be seen from the images containing the metric reports, the results obtained vary from model to model. Logistic regression produced a poor result because, considering the binary classification task, the precision is slightly higher than the completely random choice. The same is true for bagging, it too is based on the logistic regression model. It probably would have made more sense to use a random forest instead of the latter, also because the precision of the decision tree has proved to be very high. The other models based on ensemble learning have generated far too precise results. This makes me speculate that it ended up in the overfitting of the test set, even if it was never used until before the final stage. In addition, in the training part, the data obtained (visible from the graphs) were not such as to suggest an overfitting situation. I have tried a multitude of solutions, for example by balancing only the test set or not applying the PCA, but without getting better results than the current one. Honestly, I failed to understand what I believe to be the theoretical error that results in the unrealistic evaluation for those "perfect" models.

A separate discussion applies to the neural network which, with the best parameters, is able to reach an accuracy of about 90%.

In the future, several models could be exploited which may prove to be more suitable for classification tasks than those adopted by me. Additional feature engineering operations could be used by also collaborating with a domain expert and possibly some samples belonging to the smaller class could be added to balance the dataset.