Machine Learning
First Homework

Luca Tirel 1702631

***Abstract* — The aim of this report is to explain the algorithm and the solutions proposed for the text binary classification problem.**

***Index Term s*— Machine Learning, Classification, Decision Tree, Random Forest, Bug Classification, NLP, Prediction.**

I. Introduction

THE aim of this report is to provide an analysis of the classification algorithm used for the bug classification problem proposed. The dataset is composed by five columns describing some lines in Assembly and in C, the specific line of the code, the program, the path. Finally, the label consists in a binary value that say if it is a bug or not. The dataset is composed of 1.000.000 rows that can be used for the training and testing. After the model is trained, we will do the predictions on a blind data set (without labels) of 10.000 rows.

My approach to the problem will focus firstly in the comparison of the performances of three different vectorizers, on five different models. Once done that, the two best performing models and the vectorizer will be chosen and improved in a tuning process. In the end the best performing one will be chosen to predict the labels on the blind set.

All the project folder is thought to be executed offline, that is the reason for not having used a developing environment that runs on runtime.

II. Preprocessing

Before the data has been used for the training purpose, the last three columns, regarding the line of the code, the function called and the program path, has not been included in the preprocessed dataset. This is because they are not adding relevant information, moreover, since we will have to deal with the vectorization of a big text dataset, this improvement helps reducing the dimension of the processed data.

III. Vectorization

Three different models for the vectorization have been tested. They are the Count vectorizer, the Hash vectorizer and the Tfidf vectorizer.
The Hash and Count vectorizer convert a collection of text data into a matrix of token occurrences, the difference between them is that hash does not store the resulting vocabulary.
Tfidf instead, not only focus on the frequency of words present in the corpus, but also provides the importance of them, deleting the ones that are less important and making the model less complex.

The hash vectorizer has been tuned to reduce the dimension of the input data, by lowering the parameter *n_features.*
The accuracy of the models for learning proposed, that will be discussed in the next chapter in details, are listed in this tables, depending on the vectorizer chosen.

Learning Model	**Accuracy**
Decision Tree	0.9155
Logistic Regression	0.5032
Multinomial Naïve Bayes	0.5044
Random Forest	0.9471
Dummy Classifier	0.4970

Fig.1 Count Vectorizer

Learning Model	**Accuracy**
Decision Tree	0.9179
Logistic Regression	0.4959
Multinomial Naïve Bayes	0.4948
Random Forest	0.9484
Dummy Classifier	0.4970

Fig.2 Hash Vectorizer

Learning Model	**Accuracy**
Decision Tree	0.9009
Logistic Regression	0.4954
Multinomial Naïve Bayes	0.4945
Random Forest	0.9422
Dummy Classifier	0.4970

Fig. 3 Tfidf Vectorizer

Of course, these values are not fixed and depends randomly on many parameters, like the random seed, but can be used to deduce the best vectorizer to use at a first sight.

The number of the columns of the vectorized input are listed in the following table:

Vectorizer	Dimension
CountVectorizer	1652
HashVectorizer	8192
TfidfVectorizer	1652

From these numbers we choose to use the Hash vectorizer, with n_features fixed to a non-default value.

The data has been normalized after the vectorization.

IV. Model Description

Before going on in the analysis, I choose to focus on these two methods for classifications and explore them comparing solutions obtained.

A. Decision Tree

The decision tree is a model used in classification problems. It consists of some internal nodes, that test a specific feature. Each branch represents an outcome of the test, and each leaf node represent a class label predicted.

The three sometimes may be trained too well on the data, this phenomenon is named overfitting and that happens when we found a new hypothesis for classification that produce a smaller sampling error, but the real error of this new hypothesis increase. To avoid this, we can prune the tree after it is built, by removing some branches and substituting them with some leaf nodes.

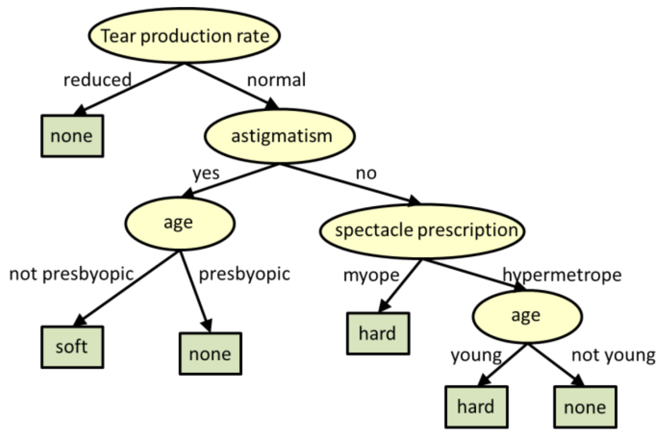

Fig.4 Example of Decision Tree

B. Random Forest

This method is similar to the decision tree, but it generates a set of decision tree with some random criteria and integrates their values into a final result. This method is very powerful cause it is based on a large number of trees with low correlation between each other. The large number of trees helps each other to protect the model from the error of a single tree. In general, this kind of structure is less likely to overfit.

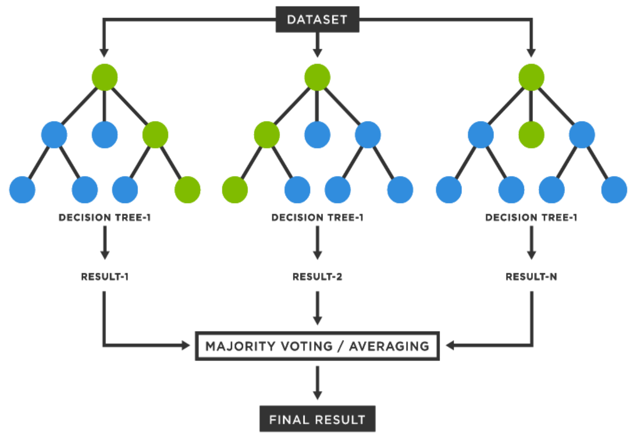

Fig. 5 Random Forest

V. Model Tuning

The hyperparameters of two model proposed are then tuned using a Grid Search Cross Validation procedure.
For the random forest this process does not outcome any improving in the accuracy of the model, instead for the decision tree, we can achieve 92% of accuracy and an optimal set of hyperparameters.

To do that, I had to define a subset of the hyperparameters that I am interested to vary and the values that the search algorithm will test. On default, the optimization is with respect to the accuracy, but I tried to search for best candidates' solutions optimizing with respect to the recall and F1-score.

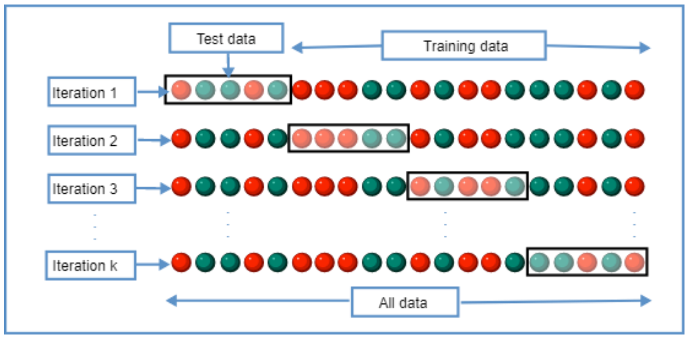

Fig.6 Grid Search Cross Validation procedure

An alternative approach to this is performing a search on the given set of hyperparameters by a random criterion. This procedure is called random search cross validation.

VI. Model Performances

In this chapter the performances of the models proposed in terms of accuracy, recall and F1-score are analyzed, by varying some hyperparameters, and keeping the others fixed.

In the following figures are shown the Confusion Matrices of two method proposed.

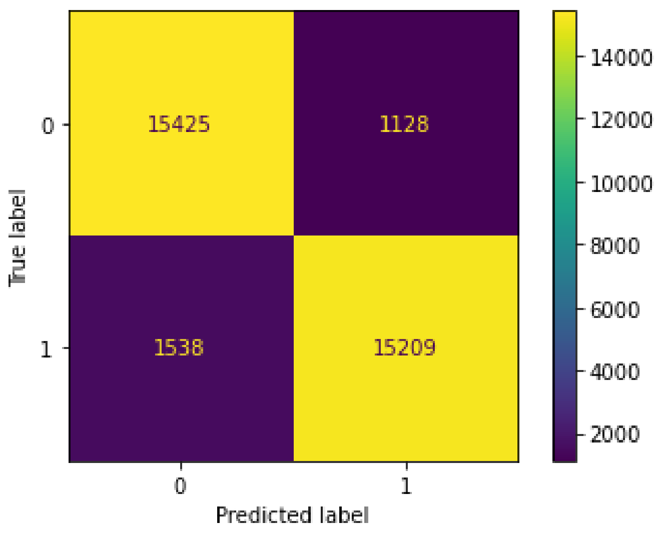

Fig.7 Confusion Matrix for Decision Tree

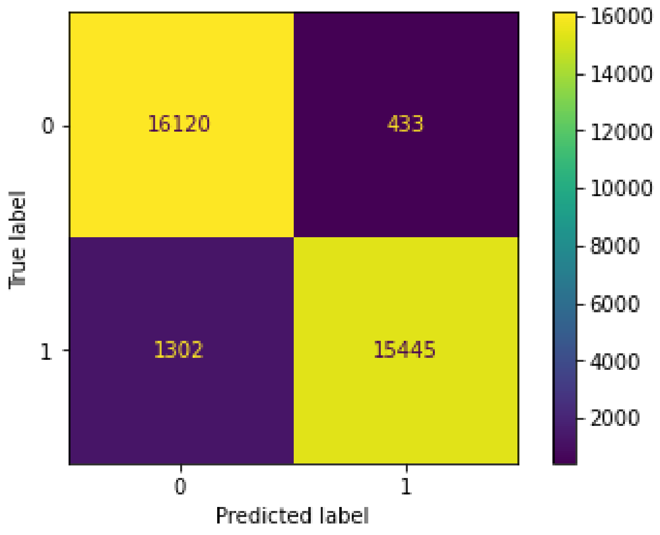

Fig.8 Confusion Matrix for Random Forest

1. Decision Tree

a) max_depth

This hyperparameter regulate the maximum depth of the tree, on default the tree has no maximum depth.

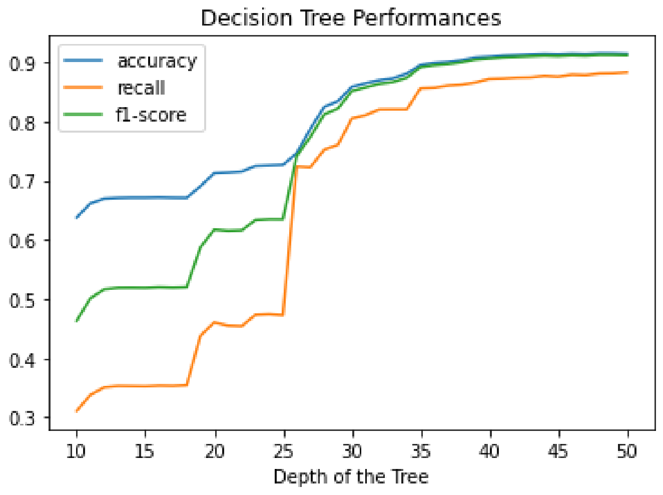

Fig.9 Scores of Decision Tree varying max_depth

From the plot we can say that after a certain depth the performances of the model are not affected anymore.

b) min_samples_split

This hyperparameter regulate the number of samples needed to split an internal node, default value is two samples.

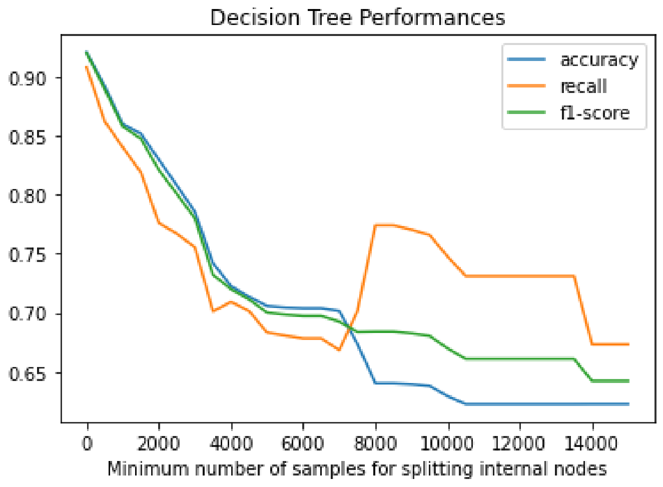

Fig.10 Scores of Decision Tree varying min_samples_split

From this plot we can say that the lower number of samples is asked, the better in term of performances of the decision tree model.

c) min_samples_leaf

This hyperparameter regulate the minimum number of samples required to be at a leaf node. Its default value is only one sample.

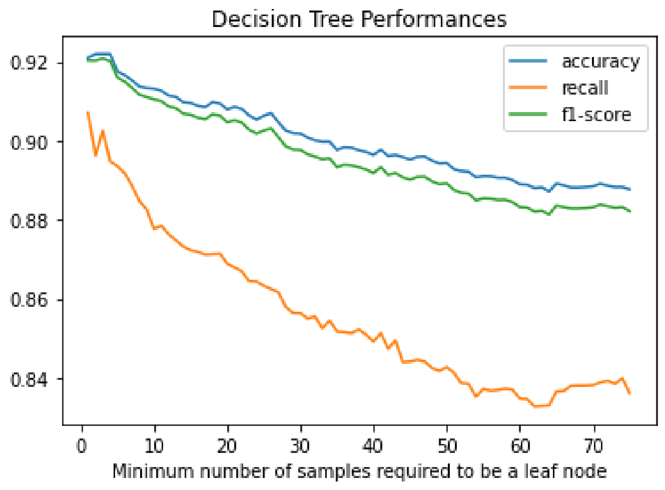

Fig.11 Scores of Decision Tree varying min_samples_leaf

From this plot we can say that we have better performances when this parameter is very low.

2. Random Forest

a) n_estimators

This parameter regulates the number of trees in the forest, we expect the lower it is, the lowest accuracy, but reduce this value can speed up computations.

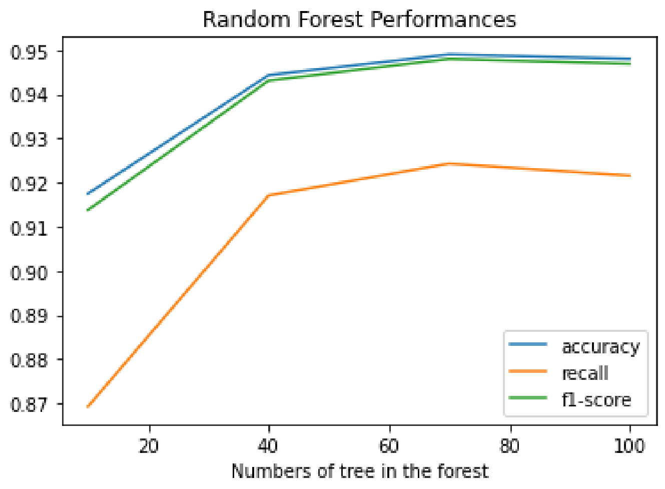

Fig.12 Scores of Random Forest varying n_estimators

b) min_sample_split

This parameter regulates the minimum number of samples required to split an internal node, the same in the decision tree.

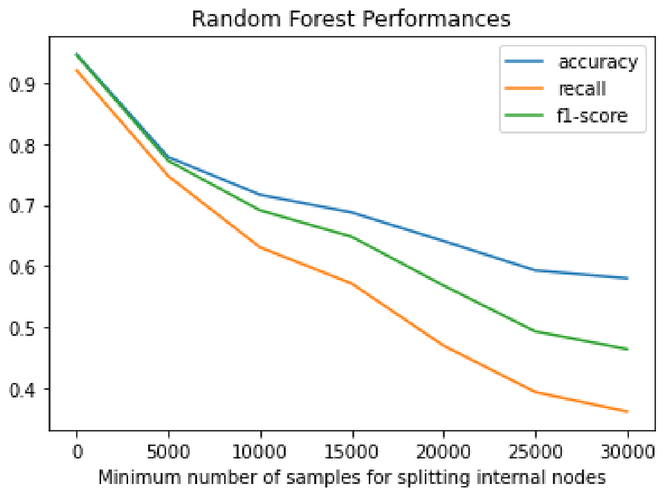

Fig.13 Scores of Random Forest varying min_samples_split

VII. Conclusion

Five different models and three vectorizers has been compared, two models have been chosen. A grid search has been performed on the decision tree to find the optimal tuning of the parameters. After tuning, the two models proposed has been compared. The model chosen to predict on the blind set is the Random Forest Classifier, due to its increased accuracy, reaching about 95%.