Classification

Marco Nobile

May 2019

1 Introduction

The task of this project is to build 5 different classification algorithms to predict bug proneness of classes from code and NLP metrics, where we can say that a bug prone class is a class that deserves verification and validation because it could contain bugs.

2 Data extraction and pre-processing

Given the task presented above, we now have to understand how we could predict if a class is considered buggy or not. To do so we are going to parse the code contained in the *Closure* library (in particular in the *jscomp* folder), and extract the following measurements:

- Class metrics
  - number of methods as MTH
  - number of fields as FLD
  - number of public methods + number of called methods as RFC
  - number of implemented interfaces as INT
- Method metrics (max over all methods)
  - max(number of statements) as SZ
  - max(number of conditional and loop statements) as CPX
  - max(number of exceptions in throws clause) as EX
  - max(number of return points) as RET
- NLP metrics
  - number of block comments as BCM
  - average length of method names as NML
  - number of words (longest alphanumeric substrings) in block comments as WRD
  - number of words in comments over the number of statements as DCM

After gathering all this information, we obtain a dataframe of shape (281, 14) indexed by the name of the class, and as feature coordinates the relative measurements for that class. In the following table I'm going to present the mean and the standard deviation for each feature:

metric	MTH	FLD	RFC	INT	SZ	CPX	EX	RET
mean	12.014	6.762	75.58	0.673	19.623	6.039	0.117	3.772
std	20.792	13.457	124.255	0.681	35.725	8.726	0.364	7.401
metric	BCM	NML	WRD	DCM				
mean	13.829	13.764	324.673	17.549				
std	22.056	4.714	428.981	49.284				

Table 1: Mean and the standard deviation for each feature

I've also decided to report the mean and the standard deviation for each feature, for the two different levels of buggy:

target class	MTH mean	MTH std	FLD mean	FLD std	RFC mean	RFC std	INT mean	INT std
buggy = 0	8.746	11.018	5.068	9.052	47.683	67.536	0.624	0.642
buggy = 1	20.829	34.290	11.329	20.597	150.829	193.311	0.803	0.766
target class	SZ mean	SZ std	CPX mean	CPX std	EX mean	EX std	RET mean	RET std
buggy = 0	14.380	22.273	4.507	5.810	0.117	0.351	2.775610	4.455
buggy = 1	33.763	56.016	10.171	12.990	0.118	0.399	6.460526	11.852
target class	BCM mean	BCM std	NML mean	NML std	WRD mean	WRD std	DCM mean	DCM std
buggy = 0	0.117	0.351	2.776	4.455	236.0	310.924	17.328	50.0
buggy = 1	0.118	0.399	6.460	11.852	563.855	587.037	18.146	47.598

Table 2: Mean and the standard deviation for each feature for the 2 levels of target

we can see that for the majority of the features, the means are quite different. We can also present the frequencies and percentages of the target class:

Target value	frequency	%
0	205	72.954%
1	76	27.046%

Figure 1: Target frequency

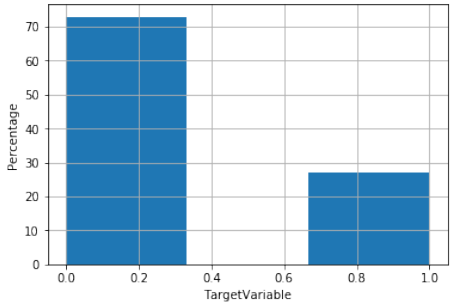

Figure 2: Hist of percentages for the target variable

3 Classifiers

Before the training process of the classifiers I have splitted the data in training (of size (224, 12)) and test set, where the test set is the 20% of the original data (57, 12). As we have seen in the preprocessing section, our features have different scales, therefore using the sklearn StandardScaler I have standardized the data, before feeding it to the classifiers.

3.1 Decision Tree

To begin with, I have trained a decision tree, which classifies data by repeated binary decisions on data attributes. Using the DecisionTreeClassifier form the sklearn.tree library, I have applied a grid search using the function GridSearchCV from sklearn.model_selection with a cross validation of size 10 and parallelizing the jobs to improve performace. I have used this technique to tune the hyperparameters of the tree and all the other models that I'm going to describe (except GaussianNB).
For the grid search I have decided to explore the following options:

- max_depth ranging between 1 and 10
- min_samples_leaf ranging between 5 and 50
- criterion gini or entropy
- max_features between sqrt or log2

After running the grid search with a cross validation of 10 splits and evaluating the models on the f1 measure, I have obtained that the best combination of parameter is:

- criterion= entropy
- max_depth = 5
- max_features = sqrt
- min_samples_leaf = 13

During the process of the grid search I had the following warning 'UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.'. This happens because in the evaluation of the different hyper parameters, it happened that the model was really bad and it was not able to output any correct prediction (and thus the f1 measure is defined as ill, with a value of 0). Training on the train set and testing on the test I have obtained a precision of 0.33, recall of 0.3, f1 of 0.316. Comparing these results with the results obtained running the algorithm with the default parameters, we can notice an improvement in the performance (for the default we get: precision: 0.182, recall: 0.6, f1 0.279, with parameters: 'criterion' = gini, max_depth = None, max_features = None, 'min_samples_leaf'=1). As we can see we the grid search has used a different criterion, reduced both max_depth and min_samples_leaf and has reduced the the number of features to consider when looking for the best split. In the following table and figure I'm going to present the non normalized confusion matrix and the normalized confusion matrix obtained using the tuned model:

	true negative	true positive
predicted negative	41	6
predicted positive	7	3

Figure 3: Confusion Matrix for the decision tree model

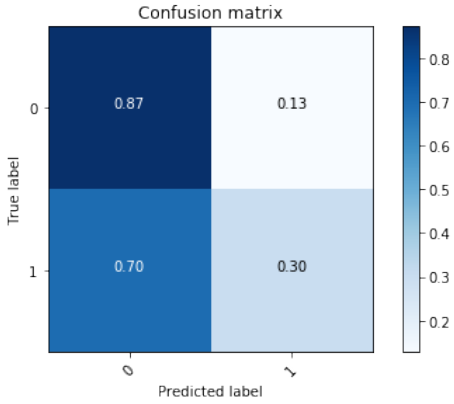

Figure 4: Normalized confusion matrix for the decision tree model

In the following figure we can see a PCA representation of the data and the points that have been wrongly classified by this model:

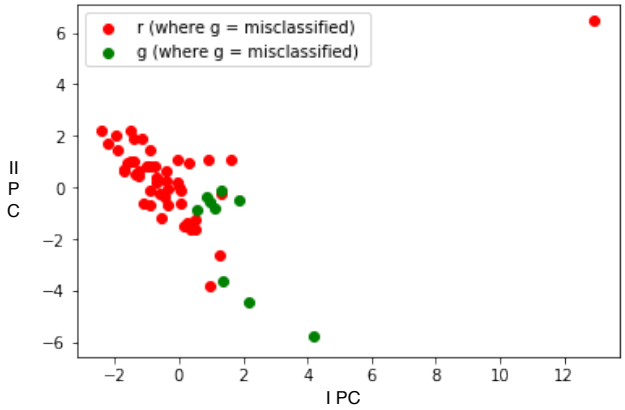

Figure 5: Scatterplot of misclassified points by the decision tree

3.2 Gaussian Naive Bayes:

Using the function GaussianNB from the library sklearn.naive_bayes we implements the Gaussian Naive Bayes algorithm for classification where the likelihood of the features is assumed to be Gaussian. I did not have the necessity to use any grid search for this algorithm because of the lack of parameters that it takes in input. I also did not have any warning from the library function. We can see the a priori probabilities on which the algorithm will run: p('buggy'=0) = 0.705 and p('buggy'=1)=0.295. Running the algorithm on the train set and testing it on the test set I have obtained a precision of 0.3, recall of 0.3 and f1 of 0.3. In the following I will present the non normalized confusion matrix and the normalized confusion matrix:

	true negative	true positive
predicted negative	40	7
predicted positive	7	3

Figure 6: Confusion Matrix for the GaussianNB model

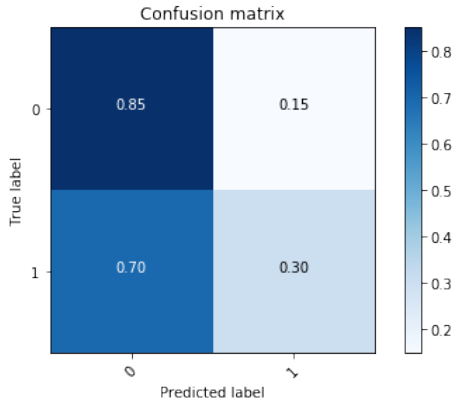

Figure 7: Normalized confusion matrix for the GaussianNB model

Finally, again I'm going to present the misclassified points:

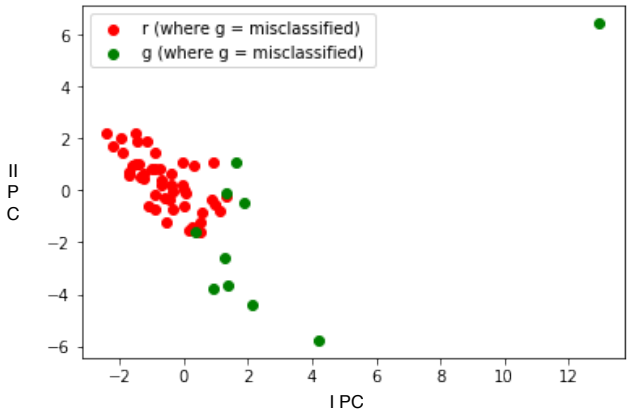

Figure 8: Scatterplot of misclassified points by the GaussianNB

3.3 Support Vectors Machines

Using the LinearSVC function from the sklearn.svm package I'm going to train a linear SVM to find the best hyperplane to separate the 2 classes that we are aiming to predict.
As shown for the decision tree model, I'm going to use a grid search to obtain the best combination of parameters to predict the target variable. The options that I've evaluated are:

- duals: True or False, where prefer dual=False when n_samples>n_features
- penaltys: l1 or l2
- loss: hinge or squared hinge as loss functions
- C: a value between 1 and 50

and the best combination of parameter that I have obtained is:

- C: 47
- dual: True
- loss: squared_hinge
- penalty: l2

During the process of the grid search I had the following warning 'UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.'. This happens because in the evaluation of the different hyper parameters, it happened that the model was really bad and it was not able to output any correct prediction (and thus the f1 measure is defined as ill, with a value of 0). With the parameters found with the grid search, I have obtained a precision of 0.307, a recall of 0.4 and an f1 score of f1 0.348. Comparing these results with a LinearSVC with the default parameters we can see that we improved the model (results of the default model: Precision: 0.2, recall: 0.2, f1 0.2, where the default parameters are: 'duals'= True, 'penalty'= l2, 'loss' = squared hinge . and C=1). As we can see the value of C of the grid search is higher (which means that the samples inside the margins are penalized more) while for the other parameters we did not have any change from the default ones.
In the following I will present the non normalized confusion matrix and the normalized confusion matrix:

	true negative	true positive
predicted negative	38	9
predicted positive	6	4

Figure 9: Confusion Matrix for the LinearSVC model

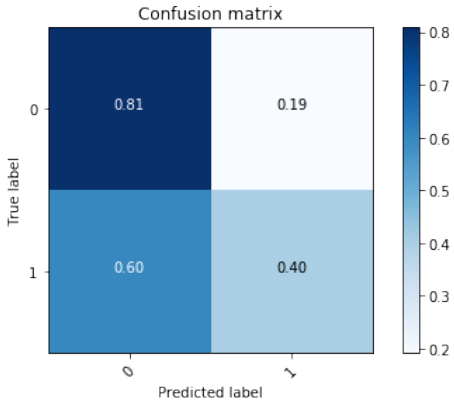

Figure 10: Normalized confusion matrix for the tuned LinearSVC model

and again I'm to present graphically the misclassified points:

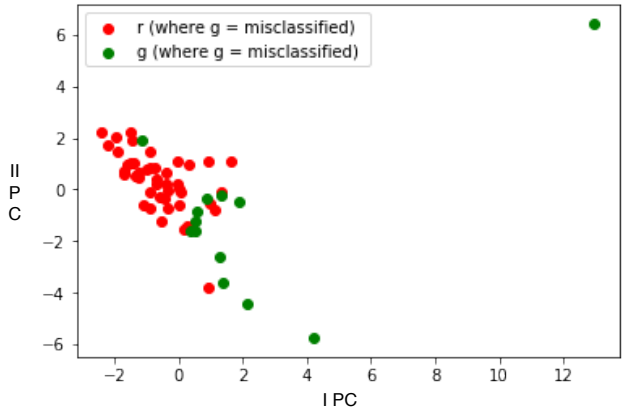

Figure 11: Scatterplot of misclassified points by the tuned LinearSVC

3.4 Multi Layer Perceptron

In this section I'm going to train a multilayer perceptron.
Using the function MLPCLassifier from sklearn.neural_network. Once again I'm going to describe the parameters I've used for the grid search:

- 'hidden_layer_sizes: a number of neuron in between 16 and 100
- activation function: either logistic, tanh or relu

I have decided to test only these parameters and to set the solver as 'lbfgs' because it can converge faster and perform better on small datasets (like ours), and to use the early stopping technique with a validation fraction of 0.1 (10% of the train data) to avoid overfitting. I did not need to pick a learning rate because of I'm not using gradient descent with lbfgs (which is an optimizer in the family of quasi-Newton methods). The parameters that I've obtained for the MLP are:

- activation: tanh
- hidden_layer_sizes: 74

during the grid search I did not have any warning. Sadly the performance for the model that I have discovered are not better the MPLClassifier with the default parameters, but during the training of the default MLPClassifier the library function issued the following ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet, which means that our optimized did not converge yet to a global optimum. Raising the parameter max_iter up to 500 resolves this problem. The precision for the default model is 0.33, the recall is 0.5 and the f1 measure is 0.4 with the following parameters: 'activation' = relu and hidden_layer_sizes = 100. As we can see the activation function defined by the grid search is different and also the number of neurons in the hidden layer is differet. Here I'm going to report the one obtained with the default model:

	true negative	true positive
predicted negative	37	10
predicted positive	5	5

Figure 12: Confusion Matrix for the MLPClassifier

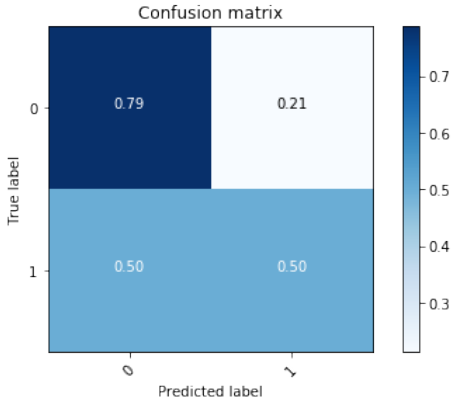

Figure 13: Normalized confusion matrix for the default MLPClassifier

In the end, again I'm going to present the scatterplot of the correctly classified points:

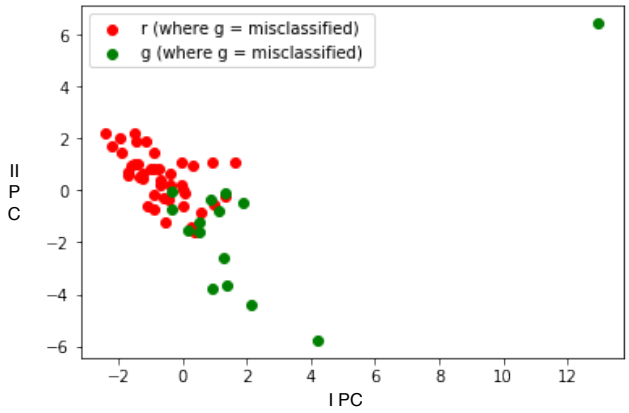

Figure 14: Scatterplot of misclassified points by the default MLPClassifier

3.5 Random Forest

Lastly I'm going to train a random forest using the function RandomForestClassifier from the package sklearn.ensamble. Once again I've used grid search to find the best parameters for our classification task:

- n_estimators: the number of trees in the forest 8, 16, 32, 64, 100
- max_depth: The maximum depth of the tree, a number between 1 to 10
- min_samples_leaf: The minimum number of samples required to split an internal node a number between 5 to 20
- criterion: the function to measure the quality of a split gini or entropy
- max_features The number of features to consider when looking for the best split, either auto, sqrt or log2

the best parameter found are:

- n_estimators: 8
- max_depth: 9
- min_samples_leaf: 7
- criterion: entropy
- max_features: sqrt

Also in this case during the grid searh I have obtained the 'UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.' Using these parameters found with the grid search, I have fitted the model and tested it on the test set, obtaining a precision of 0.25, recall: 0.3 and an f1 maesure of 0.273. Once again, sadly the results of the tuned model are less satisfactory than the one obtained with the vanilla model, which achieves a precision of 0.308, a recall of 0.4, f1 0.348, with parameters: 'n_estimators' = 10, 'max_depth' = None, 'min_samples_leaf' = 2 'criterion' = gini, 'max_features' = auto. As we can see the grid searcg has reduced the number of trees in the forest, the max depth that each tree can achieve and we increased the number of samples required before splitting (each leaf). We also changed the criterion used for splitting and the number of features to consider when looking for the best split. Given the results reported, here I'm going to present the confusion matrix for the default model:

	true negative	true positive
predicted negative	38	9
predicted positive	6	4

Figure 15: Confusion Matrix for the Random forest classifier

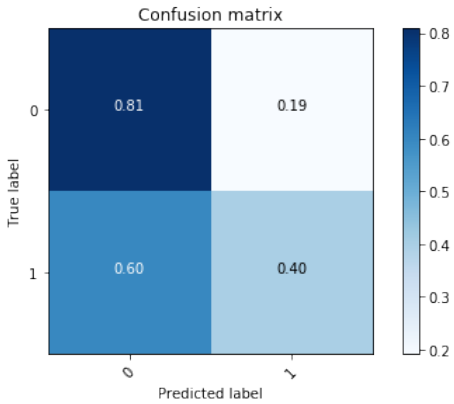

Figure 16: Normalized confusion matrix for the tuned Random forest classifier

Finally, the scatterplot of the misclassified points:

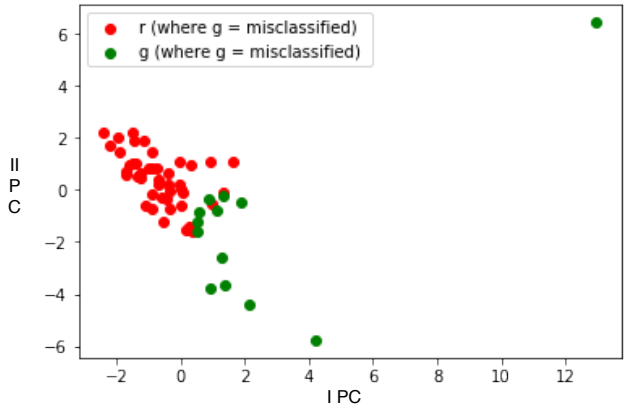

Figure 17: Scatterplot of misclassified points by the tuned Random forest classifier

3.6 Summary of precision, recall and f1 of the 5 models

Here a compact visualization of the different measurements for the 5 different models

model	precision	recall	f1 measure
Decision Tree	0.33	0.3	0.316
Gaussian NB	0.3	0.3	0.3
LinearSVM	0.307	0.4	0.348
MLPClassifier	0.33	0.5	0.4
Random Forest	0.308	0.4	0.348

Table 3: Precision, recall and f1 measure for the 5 models

Here we have the values for precision, recall and f1 measure across the different models, as we can see we have an average 0.3 of precision for all the models, we can think that this happens beacuse we are trying to predict the "buggyness" of a class with static measurements for the different classes, that might be not so informative to predict our target variable. The highest recall that we have is obtained from the MLP classifier, which, infact, has detected 5 out of the 10 true positive observations in the test set. The MLPClassifier is also the one that is more balance between precision and recall, with an f1 measure of 0.4.
This models are not for sure the top level for accuracy and prediction ability, but as stated above, this may come from different factors, as the relevance of the features or the size of the data. Surely a deeper analysis of the data or of the context could yield to an deeper and insightful analysis. I would like also to underline that these values are obtained for a single run of each classification algorithm.

4 Evaluation

Using the function DummyClassifier from the package sklearn.dummy I have build a biased classifier that I'm going to use to evaluate the results of the different models. The results of the biased classifier are:

model	precision	recall	f1 measure
DummyClassifier	0.27	1.0	0.426

Table 4: Precision, recall, f1 meausre for the biased classifier

Considering the previous section as a benchmark for model selection, in this section I'm going to present the results of the chosen models for the training process required: 5-fold cross-validation repeated 20 times to obtain 100 evaluations of the F-measures, precision/recall values of the classifiers. These values have been obtained using the function cross_validate from the package sklearn.model_selection, applied on the whole data matrix X (as required). In the following table I'm going to present the means and the standard deviations of the 100 values for each measure (n.b. the 100 values of each measure for the DummyClassifier are always the same):

model	precision mean	precision std	recall mean	recall std	f1 measure mean	f1 std
Decision Tree	0.559	0.162	0.363	0.13	0.422	0.115
Gaussian NB	0.584	0.146	0.316	0.065	0.408	0.086
LinearSVM	0.346	0.256	0.344	0.341	0.281	0.218
MLPClassifier	0.28	0.18	0.443	0.348	0.296	0.181
Random Forest	0.562	0.145	0.338	0.107	0.416	0.115
DummyClassifier	0.27	0.005	1.0	0.0	0.426	0.006

Table 5: Precision, recall, f1 meausre for 5-fold cross-validation, 20 times

Just as a reminder I will remember to the reader that the results of the MLPClassifier and the Random Forest are the one obtained with the default parameter because of the better performance obtained during the experiments presented in section 3.
As we can see using the cross_validate function we have been able to obtain a big improvement in the performance of all our models. More in particular we can say that on average we obtain at least the double in precision and f1 measure in comparison to the DummyClassifier (except for the MLPClassifier). We could not gain a lot for the recall measure because of the small amount of positive values for the target variable in the test set.

4.1 Wilcoxon's t test

In the following sections I'm going to present the results obtained running the Wilcoxon test which tests whether two dependent non-normally distributed samples were generated from populations having the same distribution:

model	precision's p-values	recall's p-values	f1 measure's p-values
Decision Tree vs GaussianNB	0.04	0.006	0.901
Decision Tree vs LinearSCV	2.768e-10	0.552	3.806e-07
Decision Tree vs MLPClassifier	9.687e-15	0.041	3.338e-08
Decision Tree vs RandomForest	0.529	0.122	0.916
Decision Tree vs DummyClassifier	4.85e-18	3.425e-18	0.839
GaussianNB vs LinearSVC	7.0e-12	0.829	6.64e-06
GaussianNB vs MLPCLassifier	8.91e-16	0.001	1.613e-06
GaussianNB vs RandomForest	0.119	0.134	0.814
GaussianNB vs DummyClassifier	2.668e-18	1.674e-18	0.081
LinearSCV vs MLPCLassifier	0.067	0.057	0.51
LinearSCV vs RandomForest	3.641e-10	0.923	1.41e-06
LinearSCV vs DummyClassifier	0.011	2.238e-17	7.1e-08
MLPClassifier vs RandomForest	2.899e-15	0.01	1.262e-07
MLPClassifier vs DummyClassifier	0.64	1.076e-16	1.71e-08
RandomForest vs DummyClassifier	6.269e-18	3.28e-18	0.815

Table 6: p-values for precision, recall and f1 measure

Considering 0.05 as threshold (and assuming every recall different from the recall of the DummyClassifier) we can see that:

- The precision distribution for Decision Tree vs GaussianNBs present significant differences
- The precision distribution for Decision Tree vs LinearSCVs present significant differences
- The precision distribution for Decision Tree vs MLPClassifiers present significant differences
- The precision distribution for Decision Tree vs DummyClassifiers present significant differences
- The precision distribution for GaussianNB vs LinearSVC presents significant differences
- The precision distribution for GaussianNB vs MLPCLassifier presents significant differences
- The precision distribution for GaussianNB vs DummyClassifier presents significant differences
- The precision distribution for LinearSCV vs RandomForestpresent presents significant differences
- The precision distribution for LinearSCV vs DummyClassifier presents significant differences
- The precision distribution for MLPClassifier vs RandomForest presents significant differences
- The precision distribution for RandomForest vs DummyClassifier presents significant differences
- The recall for Decision Tree vs GaussianNB presents significant differences
- The recall for Decision Tree vs MLPClassifier presents significant differences
- The recall for GaussianNB vs MLPCLassifier presents significant differences
- The recall for MLPClassifier vs RandomForest presents significant differences
- The f1 for Decision Tree vs LinearSCV presents significant differences
- The f1 for Decision Tree vs MLPClassifier presents significant differences
- The f1 for GaussianNB vs RandomForest presents significant differences
- The f1 for GaussianNB vs LinearSVC presents significant differences
- The f1 for GaussianNB vs MLPCLassifier presents significant differences
- The f1 for LinearSCV vs RandomForest presents significant differences
- The f1 for LinearSCV vs DummyClassifier presents significant differences
- The f1 for MLPClassifier vs RandomForest presents significant differences
- The f1 for MLPClassifier vs DummyClassifier presents significant differences

We can also try to visualize the distplots and the boxplots for the distribution of the 3 different metrics for the 5 models. Here I'm going to present as example the analysis of the Decision Tree vs GaussianNB distributions:

Here we can compare the distplots and the box plots for the f1 measure. Even if is not really clear from the distplots, we can notice that the two box plots present a similar behaviour both in the position of the mean and the size/position of the mass of the data, in fact the p-value achieved by the Wilkinson test reports a p-value of 0.901.

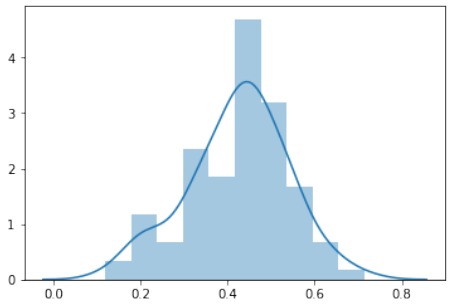

Figure 18: Distplot f1 for Decision Tree

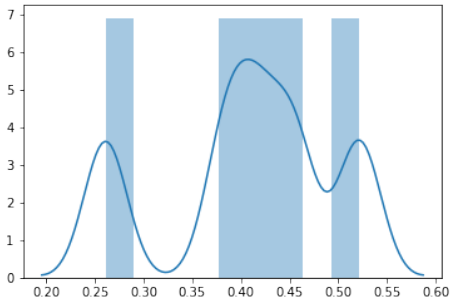

Figure 19: Distplot f1 for GaussianNB

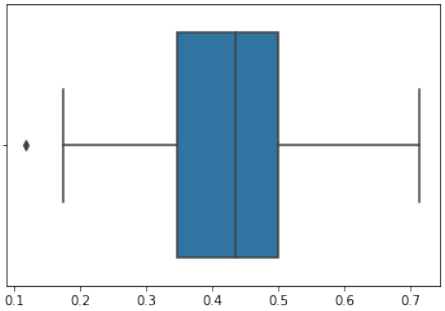

Figure 20: Boxplot f1 for DecisionTree

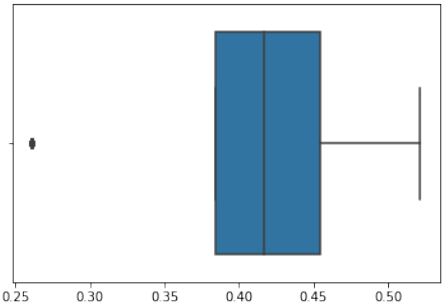

Figure 21: Boxplot f1 for GaussianNB

In the following graphics we have the comparison for the precision measure:

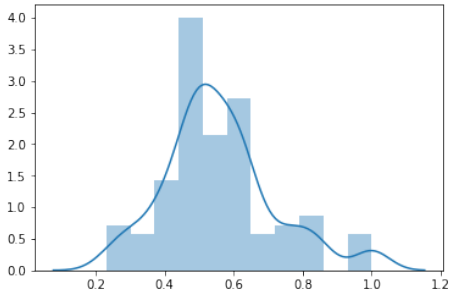

Figure 22: Distplot precision for Decision tree

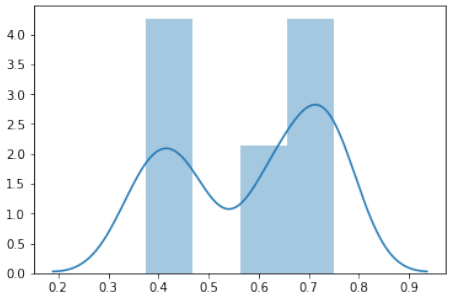

Figure 23: Distplot precision for GaussianNB

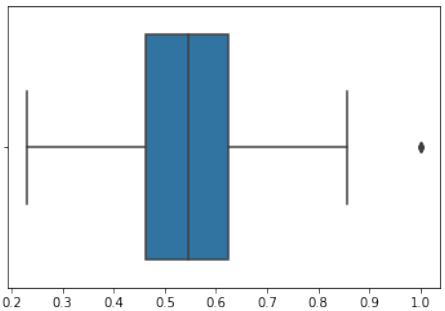

Figure 24: Boxplot precision for Decision Tree

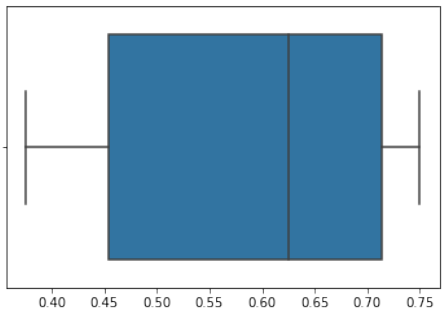

Figure 25: Boxplot precision for GaussianNB

As we can see the two distplot for the precision measure are really different,furthermore in the boxplots the mass of both the distributions are different, for this reason the Wilkinson test reports a p-value of 0.04.

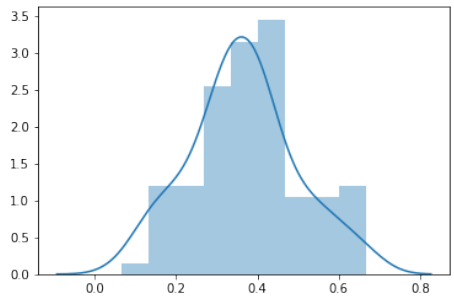

Figure 26: Distplot recall for Decision Tree

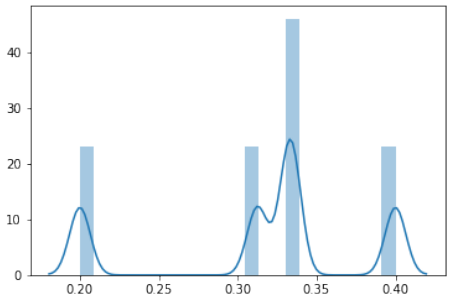

Figure 27: Distplot recall for GaussianNB

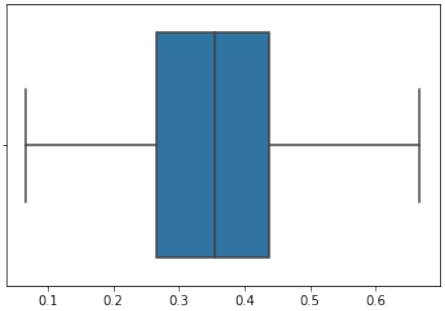

Figure 28: Boxplot recall for Decision Tree

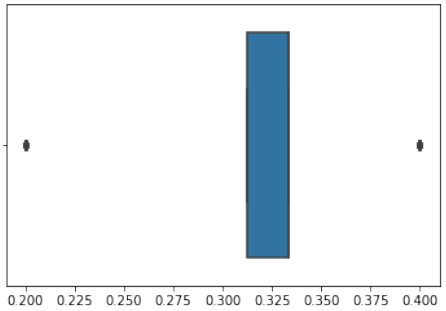

Figure 29: Boxplot recall for GaussianNB

As we can see also here the distributions and the box plots for recall are really different, in fact we achieve a p-value of 0.006.

5 Conclusions

The aim of the project was to try to build a pipeline to obtain few classifiers for bug prediction. Surely this process has been really interesting and challenging. Understanding the process of fine tuning and the different comparison in between the models has been an procedure that gave me a good insight on how the different models can behave on the same data. This said, sadly the performance of the models were not optimal, maybe more (or different metrics) data would yield to totally different results and an higher performance in bug prediction.