Project: Bug Prediction

Iva Ilcheva

Contents

Part I
Data pre-processing

1 Feature vectors

1.1 Code

1.1.1 Extract feature vectors

Numbers 3. to 6. are executed for a particular class_node. Numbers 7. to 11. are executed for a particular method_node.

1. **get_java_files()** $\longrightarrow$ Retrieves all *.java files.
2. **get_class()** $\longrightarrow$ filters the tree for ClassDeclaration and retrieves a class node by given class name.
3. **num_methods()** $\longrightarrow$ By given class node, filters tree for MethodDeclaration to record #methods. Then it does an additional filtration for MethonInvocation with the goal to retrieve #called_methods. It also invokes all method metrics related functions and returns the max out of them.
4. **num_variables()** counts for all occurrences of FieldDeclaration.
5. **num_public_methods()** counts for all occurrences of either ConstructorDeclaration or MethodDeclaration, where the member modifier is set to 'public'.
6. **num_interfaces()** $\longrightarrow$ Counts the number of node.interfaces if there are any, otherwise it returns 0.
7. **num_statements()** $\longrightarrow$ Filters the method node for Statement and returns the count of all of them except for BlockStatement.
8. **num_conditional()** $\longrightarrow$ Filters the method node for Statement and returns the count of all IfStatement and SwitchStatement with the exception of BlockStatement.
9. **num_loop()** $\longrightarrow$ Filters the method node for Statement and returns the count of all WhileStatement and ForStatement with the exception of BlockStatement.
10. **num_throws()** $\longrightarrow$ Filters the method node for Statement and returns the count of all ThrowStatement with the exception of BlockStatement.
11. **num_returns()** $\longrightarrow$ Filters the method node for Statement and returns the count of all ReturnStatement with the exception of BlockStatement.

12. **num_comments_and_words()** ⟶ Filters the class node for Documented and if there is documentation it counts that as a comment and computes the length of the comment without the white spaces.

13. **num_comments_and_words()** ⟶ For a given class' method names finds the average length of all.

14. **get_buggy_classes()** ⟶ Retrieves the names of all modified classes in Closure.

15. **is_buggy_class()** ⟶ For a given class node performs a check if it is part of the array of modified classes' names and returns 1 if that is the case and 0 otherwise.

16. At the end of the file the whole array of files is traversed and all the metrics are computed based on the methods above and appended to a dictionary, from which the feature vectors Dataframe is generated and recorded into a *.csv file.

1.2 Descriptive data

There are 280 bug prone classes in total. From the tables below we can conclude that the biggest portion of code belongs to the NLP metrics. In second place come the class metrics and third smallest portion belongs to the method metrics.

1.2.1 Class metrics

#Methods

count	280.000000
mean	11.500000
std	18.954766
min	0.000000
25%	3.000000
50%	6.000000
75%	12.250000
max	209.000000

#Fields

count	280.000000
mean	6.739286
std	13.476113
min	0.000000
25%	2.000000
50%	4.000000
75%	8.000000
max	167.000000

#Public_methods + Called_methods

count	280.000000
mean	73.203571
std	115.461011
min	0.000000
25%	14.750000
50%	30.000000
75%	80.000000
max	759.000000

#Implemented_interfaces

count	280.000000
mean	0.675000
std	0.681094
min	0.000000
25%	0.000000
50%	1.000000
75%	1.000000
max	3.000000

Out of all class metrics the **public and called methods** have the highest amount on average of about **73**, followed by **methods - 12** and **fields - 7**. The **implemented interfaces** are quite rare occurrence below **1**. Max to min observed values for all four metrics vary tremendously from 0 to 209 for methods, to 167 for fields, 759 for public and called methods and only 3 for implemented interfaces.

1.2.2 Method metrics (max over all methods)

#Statements	
count	280.000000
mean	19.539286
std	35.761935
min	0.000000
25%	5.000000
50%	11.000000
75%	20.250000
max	347.000000

#Conditional + #Loop	
count	280.000000
mean	6.360714
std	8.888107
min	0.000000
25%	1.000000
50%	4.000000
75%	9.000000
max	99.000000

#Exceptions_in_throws_clause	
count	280.000000
mean	0.392857
std	1.017386
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	9.000000

#Return_points	
count	280.000000
mean	3.635714
std	7.050977
min	0.000000
25%	1.000000
50%	2.000000
75%	4.000000
max	86.000000

Out of all method metrics the **statements** have the highest amount on average of about **20**, followed by **conditionals + loops - 6** and **Return points - 4**. The **Exceptions in throws clause** are quite rare occurrence below **1**. Max to min observed values for all four metrics vary tremendously from 0 to 347 for statements, to 99 for conditionals + loops, 86 for return points and with the exception of only 9 for exceptions in throws clause.

1.2.3 NLP Metrics

#Block_comments	
count	280.000000
mean	13.375000
std	20.737446
min	1.000000
25%	3.000000
50%	6.500000
75%	16.250000
max	221.000000

#Avg_lenght_of_method_names	
count	280.000000
mean	13.752607
std	4.719337
min	0.000000
25%	10.310000
50%	13.710000
75%	17.012500
max	28.000000314.642857

#Words	
count	280.000000
mean	314.642857
std	395.368483
min	2.000000
25%	70.000000
50%	190.000000
75%	377.000000
max	2694.000000

#Words_in_comments/#Statements	
count	280.000000
mean	38.714766
std	89.081483
min	0.000000
25%	6.241075
50%	15.612500
75%	32.683775
max	950.000000

Out of all method metrics the **words** have the highest amount on average of about **314**, followed by **words in comments/statements - 39**, **avg length of method names - 14** and **block comments - 13**. Max to min observed values for all four metrics vary tremendously from 0-2 to 2694 for words , to 950 for words in comments/statements, 221 for block comments and 28 for avg length of method names.

1.2.4 Buggy

count	280.000000
mean	0.228571
std	0.420664
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

The **buggy** descriptive statistics inform that about less than **1/4** of all the classes are considered buggy.

Part II
Classifiers

2 Classifier configurations

Each classifier was trained twice: **default** and **fine-tuned**. For each of them there will be a reasoning of the fine-tuned parameters. All non-mentioned parameters are left default.

2.1 Default

2.1.1 Decision Tree

The criterion is set to **gini**. The splitter is **best**. **Presort** is disabled. Max depth, max features, max leaf nodes, max impurity decrease and class weights are all None. Min samples split is 2. Min samples leaf is 1. Min weight fraction leaf is 0.0. Min impurity decrease is 0.0.

2.1.2 Naive Bayes

The smoothing is **1e-9**. The prior probabilities are not set.

2.1.3 Support Vector Machine

Penalty is set to **l2** which for dense vectors.**Random_state** and class weight are not defined. Loss is set to squared hinge. Dual and fit intercept are enabled. Tolerance is 0.0001. Penalty parameter C is 1.0. Multi class is one vs rest. Intercept calling is 1. Verbose is set to 0. Max iterations are 1000.

2.1.4 Multi-Layer Perceptron

The **hidden_layer_sizes** is 100. **Solver** is set to **adam**. **Activation** function used is **relu**. The **learning_rate** is **constant** and 0.001. Batch size is auto. Alpha (l2 penalty) is 0.0001. Power t is 0.5. Max iterations are 200. Shuffling and nesterovs momentum are enabled. Random state, warm start, early stopping and verbose are disabled. Tolerance is 0.0001 and momentum is 0.9. Validation fraction is 10%. Exponential decay rate for estimates of first moment vector (beta 1) is 0.9 and the second moment vector (beta 2) is 0.999. Numerical stability (epsilon) is 1e-08. Number of epochs allowed with no change is 10.

2.1.5 Random Forest

The criterion is set to **gini**. The **oob_score**, **warm_start**, max depth, max leaf nodes, n jobs, random state, min impurity split, warm start and verbose are all disabled. Bootstrap is enabled. N estimators are 100. Min samples split is 2, min samples leaf is

1 and min weight fraction leaf is 0.0. Max features are automatically computed. Min impurity decrease is 0.0.

2.2 Fine-tuned

2.2.1 Decision Tree

The criterion is set to **entropy** for information gain. The splitter is **random** and it chooses the best random split. **Presort** is enabled with the hopes to possibly speed up the finding of best splits in fitting.

2.2.2 Naive Bayes

Experimented with less smoothing by setting **var_smoothing** to **1e-3**. It represents the portion of the largest variance of all features and is added to all other variances to aid calculation stability.

2.2.3 Support Vector Machine

Penalty is set to **l1** which leads to sparse vectors. Also **random_state** is 1 as a tuned seed for data shuffling.

2.2.4 Multi-Layer Perceptron

The **hidden_layer_sizes** is increased to 1000. **Solver** is set to **stochastic gradient descent**. The aim is to maximize the accuracy. **Activation** function used is **tanh**. The **learning_rate** is changed to **adaptive** to react early on poorer loss evaluations.

2.2.5 Random Forest

The criterion is set to **entropy** for information gain. The **oob_score** is enabled so that it uses out-of-bag samples to estimate the generalization accuracy. **Warm_start** is enabled in order to use the previous call to fit and not create a whole new forest.

3 Precision and recall

3.1 Report

All of these **fine-tunings** performed overall worse than the **default** versions of the classifiers. For this section I will be comparing the **precision_recall_fscore_support** outputs for both cases and for each classifier.

Table 1: Decision Tree

Version	Precision	Recall	FScore
Default	0.4666666666666667	0.5	0.4827586206896552
Fine-tuned	0.2631578947368421	0.35714285714285715	0.30303030303030304

The **Decision Tree** fine-tuned case performed almost third as worse over all performance measures.

Table 2: Naive Bayes

Version	Precision	Recall	FScore
Default	0.4	0.14285714285714285	0.21052631578947364
Fine-tuned	0.3333333333333333	0.07142857142857142	0.11764705882352941

Similar to the **Decision Tree**, **Naive Bayes** fine-tuned case performed worse over all performance measures but around half as the default. Only exception is the precision which in both cases varies only by 0.7.

Table 3: Support Vector Machine

Version	Precision	Recall	FScore
Default	0.37142857142857144	0.9285714285714286	0.5306122448979592
Fine-tuned	0.45454545454545453	0.7142857142857143	0.5555555555555556

Support Vector Machine scores show that actually here the fine-tuned case performed slightly better with the exception only of the recall which is 0.2 points than the default.

Table 4: Multi-Layer Perceptron

Version	Precision	Recall	FScore
Default	0.5	0.21428571428571427	0.3
Fine-tuned	1.0	0.07142857142857142	0.13333333333333333

Similar to the first classifier fine-tuned case of **Multi-Layer Perceptron** performed worse over all performance measures by the same proportion. Only exception is the recall which increased twice, resulting in the highest possible amount of 1 on the dot.

Table 5: Random Forest

Version	Precision	Recall	FScore
Default	0.4	0.14285714285714285	0.21052631578947364
Fine-tuned	0.25	0.07142857142857142	0.11111111111111112

Interesting observation is that **Decision Tree** and **Random Forest** have the same scores for the default cases. However, they differ in the fine-tuned versions. **Random**

Forest performs similarly to **Naive Bayes** when its own two versions are compared - twice as bad as the default in favour of the fine-tuned.

3.2 Errors

There is a **convergence warning** for all classifiers, except biased. The number of iterations is not enough for any of them to converge. **Support Vector Machine** and **Multi-Layer Perceptron** output that precision and f-score are ill-defined and being set to 0.0 due to failure in prediction of samples.

Part III
Evaluation

4 Descriptive statistics and box plots

4.1 Descriptive statistics

4.1.1 Decision Tree

Table 6: Decision Tree

Metric	Mean	Std
F1	0.35858539739704803	0.07347222130228126
Precision	0.34283034006756913	0.07448407973156956
Recall	0.38173076923076926	0.0856213271910333

The center of the distribution of the cross-validation test **f1** is **0.36**, **precision** is **0.34** and **recall** is **0.38**. From the CV (standart deviation/mean) low value $\sim 0.2 < 1$ of the three we can conclude that the data points tend to be spread around and close to the mean.

4.1.2 Naive Bayes

Table 7: Naive Bayes

Metric	Mean	Std
F1	0.40344862155388467	0.08073256716520394
Precision	0.5484432234432234	0.15470727518375002
Recall	0.32820512820512826	0.05710527551620535

The center of the distribution of the cross-validation test **f1** is **0.40**, **precision** is **0.55** and **recall** is **0.33**. From the CV (standart deviation/mean) low value $\sim 0.29 < 1$ of the three we can conclude that the data points tend to be spread around and close to the mean.

4.1.3 Support Vector Machine

Table 8: Support Vector Machine

Metric	Mean	Std
F1	0.2487086553142601	0.2010249431161487
Precision	0.3841031217457688	0.32356947128192376
Recall	0.2791025641025641	0.2975000207183458

The center of the distribution of the cross-validation test **f1** is **0.25**, **precision** is **0.38** and **recall** is **0.28**. From the CV (standart deviation/mean) low value $\sim 1.05 > 1$ of the three we can conclude that the data points are spread out over a wider range of values.

4.1.4 Multi-Layer Perceptron

Table 9: Multi-Layer Perceptron

Metric	Mean	Std
F1	0.31943386895853065	0.11837255560242206
Precision	0.40671767866766423	0.14146484795415173
Recall	0.31750000000000006	0.19098359329005962

The center of the distribution of the cross-validation test **f1** is **0.32**, **precision** is **0.41** and **recall** is **0.32**. From the CV (standart deviation/mean) low value $\sim 0.6 < 1$ of the three we can conclude that the data points tend to be spread around and close to the mean.

4.1.5 Random Forest

Table 10: Random Forest

Metric	Mean	Std
F1	0.3238214527347587	0.12007835279197528
Precision	0.49799494949494943	0.18322619389509495
Recall	0.25012820512820516	0.10374710375938954

The center of the distribution of the cross-validation test **f1** is **0.33**, **precision** is **0.5** and **recall** is **0.25**. From the CV (standart deviation/mean) low value $\sim 0.41 < 1$ of the three we can conclude that the data points tend to be spread around and close to the mean.

4.2 Boxplots

4.2.1 Decision Tree

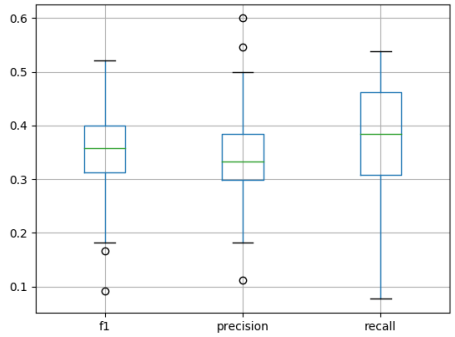

Figure 1: Decision Tree

The distribution of **f1** indicates that the data is normally distributed with the median lying in the middle of Q1 and Q3 and a few outliers below the bottom whisker. The distribution of **precision** requests similar distribution but the median tends to lie closer to Q1 and there are more outliers above the top whisker and only 1 below the bottom one. The distribution of **recall** states that the data is top-skewed (not normally distributed) and closer to the top whisker with the median lying just in the middle of Q1 and Q3. There are no outliers.

4.2.2 Naive Bayes

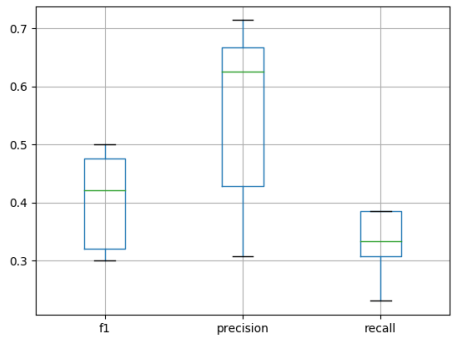

Figure 2: Naive Bayes

The distribution of **f1** indicates that the data is normally distributed with the median lying closer to Q3 and no outliers. The distribution of **precision** states that the data is top-skewed (not normally distributed) and closer to the top whisker with the median lying very close to Q3. The distribution of **recall** states that the data is top-skewed (not normally distributed) and overlapping the top whisker with the median lying closer to Q1. There are no outliers.

4.2.3 Support Vector Machine

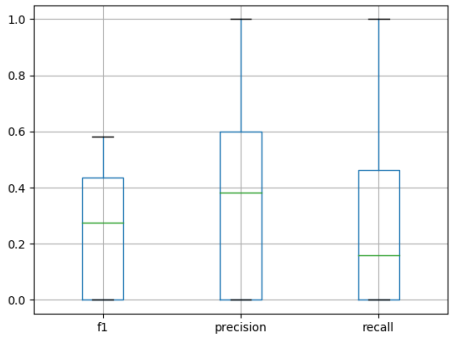

Figure 3: Support Vector Machine

The distribution of **f1** indicates that the data is bottom-skewed (not normally distributed) and overlapping the bottom whisker with the median lying very close to Q3. The distribution of **precision** and the distribution of **recall** are the same as **f1**.

4.2.4 Multi-Layer Perceptron

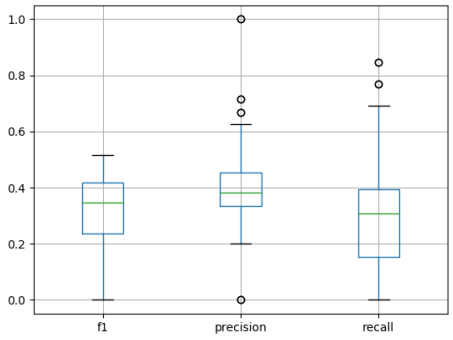

Figure 4: Multi-Layer Perceptron

The distribution of **f1** indicates that the data is top-skewed (not normally distributed) with the median lying closer to Q3 and no outliers. The distribution of **precision** states that the data is relatively normally distributed median lying slightly closer to Q1. There are a lot of outlier below and above the two whiskers. The distribution of **recall** states that the data is bottom-skewed (not normally distributed) with the median lying closer to Q3. There are outliers above the top whisker.

4.2.5 Random Forest

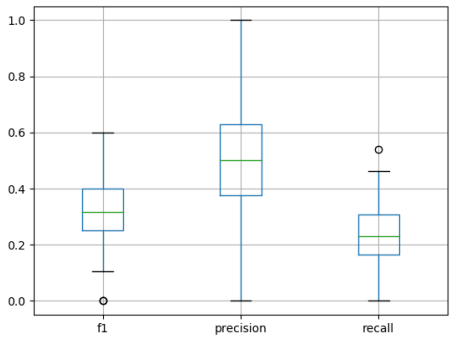

Figure 5: Random Forest

The distribution of **f1** indicates that the data is relatively normally distributed with the median lying just in the middle of Q1 and Q3 and an outlier below the bottom whisker. The distribution of **precision** states that the data is normally distributed with median lying in the center between Q1 and Q3. There are no outliers. The distribution of **recall** suggests that the data is normally distributed with median lying in the center between Q1 and Q3. There is an outlier above the top whisker.

4.3 Comparison with biased classifier

Table 11: Biased

Metric	Mean	Std
F1	0.37201446185222947	0.007210724196964231
Precision	0.22853611300979726	0.005412103637755966
Recall	1.0	0.0

The center of the distribution of the cross-validation test **f1** is **0.37**, **precision** is **0.23** and **recall** is **1** (as it is biased). From the CV (standart deviation/mean) low value $\sim$ 0.02 to 0.0 $<$ 1 of the three we can conclude that the data points tend to be spread

around and close to the mean. The recorded CV from the biased compared to the other classifiers is the lowest one with the elements lying closest to the mean.

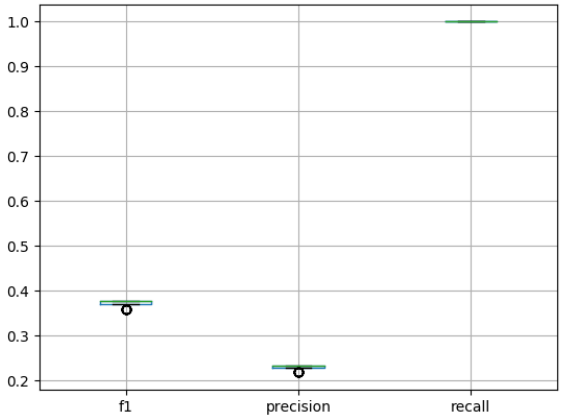

Figure 6: Biased

The the three variables are distributed in a very strange way. There seem to be indentical points, matching with the median, whiskers and quartiles. No variety in the data whatsoever which is completely opposite of what was observed for the other classifiers.

5 Wilcoxon's p values

$$classifiers\_dict = ($$

$$\textbf{CLF} : DecisionTreeClassifier,$$

$$\textbf{GNB} : GaussianNB,$$

$$\textbf{LSVC} : LinearSVC,$$

$$\textbf{MLPC} : MLPClassifier,$$

$$\textbf{RFC} : RandomForestClassifier,$$

$$\textbf{BIASED} : BiasedClassifier$$

$$)$$

5.1 Statistical significance of observed evaluation metrics difference between classifiers

Table 12: Wilcoxon's p values between classifiers

Pair	F1	Precision	Recall
CLF - RFC	0.04194404264075937	1.2184334767785097e-12	4.4425602408683077e-14
GNB - LSVC	1.3550349094910103e-05	6.204561022378865e-09	0.0999147073428363
MLPC - LSVC	0.38822237253079583	0.06416607070890841	0.29300786386291866
GNB - RFC	9.83054486509396e-07	0.00042557496967490533	1.11680638317677e-08
CLF - MLPC	0.010048333144880755	0.0003744490939098602	1.6028209626420805e-05

5.1.1 Decision Tree vs Random Forest

For *F1* we can reject the null hypothesis that that the compared distributions are identical. However, for both *precision* and *recall* we can't reject it.

5.1.2 Naive Bayes vs Support Vector Machine

For *F1*, *precision* and *recall* we can't reject the null hypothesis that that the compared distributions are identical.

5.1.3 Multi-Layer Perceptron vs Support Vector Machine

For *F1*, *precision* and *recall* we can reject the null hypothesis that that the compared distributions are identical and conclude that they are nonidentical.

5.1.4 Naive Bayes vs Random Forest

For *F1* and *recall* we can't reject the null hypothesis that that the compared distributions are identical. However, for the *precision* we can reject it and conclude they are nonidentical.

5.1.5 Decision Tree vs Multi-Layer Perceptron

For *F1* and *precision* we can reject the null hypothesis that that the compared distributions are identical and conclude that they are nonidentical. However, for the *recall* the case is the opposite.

5.2 Statistical significance of observed evaluation metrics difference with biased classifier

Table 13: Wilcoxon's p values with biased

2nd Classifier	F1	Precision	Recall
CLF	0.0006352985451101117	1.0215266210052788e-25	1.9751486318855778e-39
GNB	0.006482257418998102	9.130700150590715e-36	9.475406335351758e-40
LSVC	0.23562337514239534	2.848325707357369e-07	4.225393249628847e-37
MLPC	0.0002598596860476937	4.349592161163476e-27	2.471988920086036e-39
RFC	0.0004466626138977573	3.1689640288550644e-29	2.088295822122853e-39

The comparisons are made between the biased classifier and each of the 2nd ones. With all of them for *F1* we can reject the null hypothesis that that the compared distributions are identical. However, for both *precision* and *recall* we can't conclude such an outcome.

5.3 Conclusion

5.3.1 F1

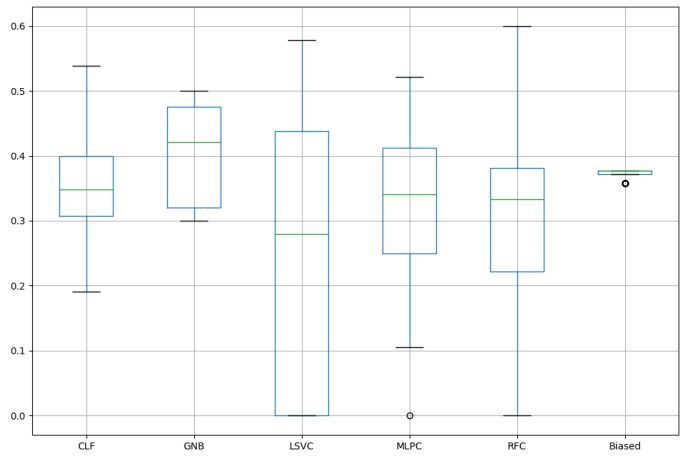

Figure 7: F1

Best performing classifier for **F1** measure is Naive Bayes.

5.3.2 Precision

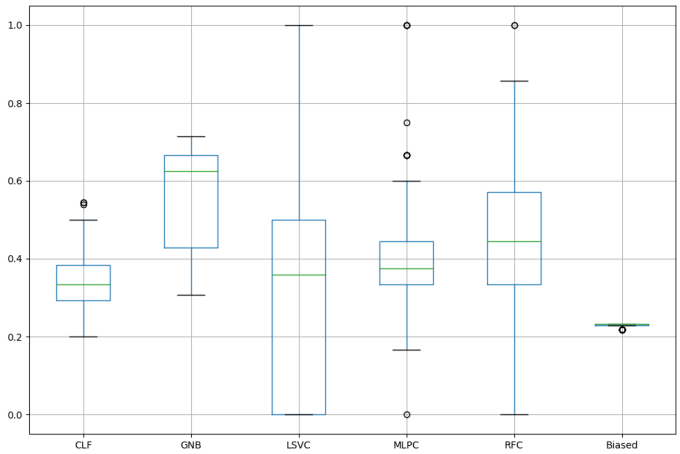

Figure 8: Precision

Best performing classifier for **precision** measure is Naive Bayes.

5.3.3 Recall

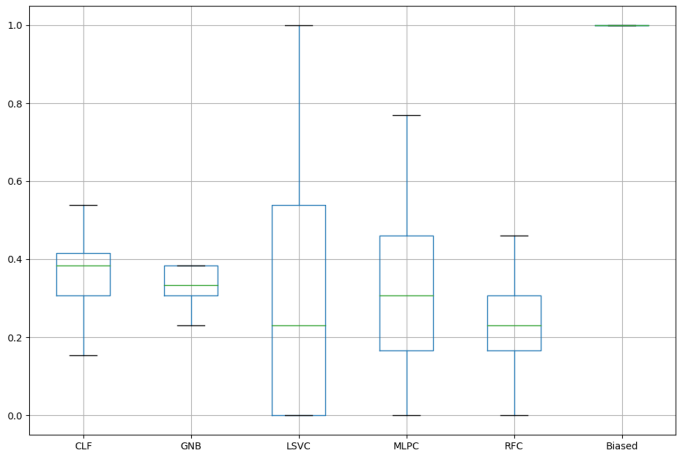

Figure 9: Recall

Best performing classifier for **recall** measure is the Biased. But since we are more interested in the non-biased classifiers, the best performace was observed with the Decision Tree.

5.3.4 Wilcoxon

F1 best p value can be seen in the row of the comparison between Decision Tree and Multi-Layer Perceptron. Same is the case for the precision p value. However, best observed p value of the recall comparisons is between Naive Bayes and Support Vector Machine.

6 Practical usefulness of the obtained classifiers in a realistic bug prediction scenario

Let's again imagine the setup of having a team of developers before and after introducing bug prediction into their daily work. Training classifiers to predict accurately bugs and using the models regularly while working on a big project with a lot of people would be beneficial. It could serve a vital role next to testing and even automated god class clustering during the development process. It would be ideal to check for any bug prone code before merging it into production at the end of each work day to detect possible pieces of it that need further testing. This would stop the team of committing buggy code that will cause overhead in the future. Developers will easily find a bug in generally less amount of code than of a few hundred lines. It might take a bit more time to go through such a procedure everyday like the god class clustering but so is testing and if done right, saves a lot of time. I am glad I learnt about this problem and this possible way of preventing it and/or fixing it. I'd definitely use this approach in my future ventures.

Part IV
Appendix

Executing **sh run.sh** in the command line in the same directory runs all scripts sequentially in the correct order to produce the desired results. Each script produces a folder with the same name and the corresponding ***.csv** files.

Listing 1: run.sh

```
#!/bin/bash
echo 'Extracting the feature vectors...'
python3 extract_feature_vectors.py
echo 'Finished!'
echo 'Training the classifiers...'
python3 train_classifiers.py
echo 'Finished!'
echo 'Computing classifier evaluations...'
sh ./evaluate-classifiers.sh
echo 'Finished!'
echo 'Performing Wilcoxon tests...'
sh ./wilcoxon.sh
echo 'Finished!'
```

- Extract and label vectors: **extract_feature_vectors.py**
- Train classifiers: **train_classifiers.py**
- Evaluate classifiers: **evaluate-classifiers.py**

NB!: *defects4j* folder has to be in the same directory.