Information Modelling & Analysis – Project 2

Claudio Maggioni

Code Repository

The code and result files, part of this submission, can be found at:

- Repository: https://github.com/infoMA2023/project-02-bug-prediction-maggicl
- Commit ID: **5f30b3b71b50b28f4c870ff114a760ece1005531**

Data Pre-Processing

I use the sources of the Closure repository that were already downloaded using the command:

```
defects4j checkout -p Closure -v 1f -w ./resources
```

and used the code in the following subfolder for the project:

```
./resources/defects4j-checkout-closure-1f/src/com/google/javascript/jscomp/
```

relative to the root folder of the repository. The resulting CSV of extracted, labelled feature vectors can be found in the repository at the path:

```
./metrics/feature_vectors_labeled.csv
```

relative to the root folder of the repository.

Unlabeled feature vectors can be computed by running the script `./extract_feature_vectors.py`. The resulting CSV of unlabeled feature vectors is located in `./metrics/feature_vectors.csv`.

Labels for feature vectors can be computed by running the script `./label_feature_vectors.py`.

Feature Vector Extraction

I extracted **291** feature vectors in total. Aggregate metrics about the extracted feature vectors, i.e. the distribution of the values of each code metric, can be found in Table 1.

Table 1: Distribution of values for each extracted code metric.

Metric	Minimum	Average	Maximum
BCM	0	13.4124	221
CPX	0	5.8247	96
DCM	0	4.8652	176.2
EX	0	0.1134	2
FLD	0	6.5773	167
INT	0	0.6667	3
MTH	0	11.6529	209
NML	0	13.5622	28
RET	0	3.6735	86
RFC	0	107.2710	882
SZ	0	18.9966	347
WRD	0	314.4740	3133

Feature Vector Labelling

After feature vectors are labeled, I determine that the dataset contains **75** buggy classes and **216** non-buggy classes.

Classifiers

In this section I explain how I define and perform training for each classifier.

Since the dataset has an unbalanced number of feature vectors of each class, in order to increase classification performance I upsample the dataset by performing sampling with replacement over the least frequent class until the number of feature vectors matches the most frequest class[1].

Other than for the `GaussianNB` (Naive Bayes) classifier, the classifiers chosen for the project offer to select hyperparameter values. In order to choose them, I perform a grid search over each classifier. The hyperparameter values I have considered in the grid search for each classifier are the following:

- For *DecisionTreeClassifier*:

Parameter	**Values**
criterion	gini, entropy
splitter	best, random

- For *SVC*:

Parameter	**Values**
kernel	linear, poly, rbf, sigmoid
gamma	scale, auto

- For *MLPClassifier*:

Parameter	**Values**
max_iter	500000
hidden_layer_sizes	$[5, 10, 15, ..., 100]$, $[15, 30, 45, 60, 75, 90]^2$, $[20, 40, 60, 80, 100]^3$
activation	identity, logistic, tanh, relu
solver	lbfgs, sgd, adam
learning_rate	constant, invscaling, adaptive

  Note that the $[...]^2$ denotes a cartesian product of the array with itself, and $[...]^3$ denotes the cartesian product of $[...]^2$ with the array (i.e. $[...]^3 = [...]^2 \times [...] = ([...] \times [...]) \times [...]$).

  Note also the high upper bound on iterations (500000). This is to allow convergence of the less optimal hyperparameter configurations and avoid `ConvergenceWarning` errors.

- For *RandomForestClassifier*:

Parameter	**Values**
criterion	gini, entropy
max_features	sqrt, log2
class_weight	balanced, balanced_subsample

[1] Upsampling due to unbalanced classes was suggested by *Michele Cattaneo*, who is attending this class.

The script `./train_classifiers.py`, according to the random seed 3735924759, performs upscaling of the dataset and the grid search training, by recording precision, accuracy, recall and the F1 score of each configuration of hyperparameters. These metrics are then collected and stored in `./models/models.csv`.

The metrics for each classifier and each hyperparameter configuration in decreasing order of accuracy are reported in the following sections.

For each classifier, I then choose the hyperparameter configuration with highest accuracy. Namely, these configurations are:

Classifier	**Hyper-parameter configuration**	**Precision**	**Accuracy**	**Recall**	**F1 Score**
DecisionTreeClassifier	`criterion`: gini, `splitter`: best	0.7885	0.8506	0.9535	0.8632
GaussianNB	–	0.8	0.6782	0.4651	0.5882
MLPClassifier	`activation`: logistic, `hidden_layer_sizes`: (60, 80, 100), `learning_rate`: constant, `max_iter`: 500000, `solver`: lbfgs	0.8958	0.9425	1	0.9451
RandomForestClassifier	`class_weight`: balanced, `criterion`: gini, `max_features`: sqrt	0.8367	0.8851	0.9535	0.8913
SVC	`gamma`: scale, `kernel`: rbf	0.7174	0.7356	0.7674	0.7416

Decision Tree (DT)

criterion	splitter	precision	accuracy	recall	f1
gini	best	0.788462	0.850575	0.953488	0.863158
gini	random	0.784314	0.83908	0.930233	0.851064
entropy	random	0.736842	0.816092	0.976744	0.84
entropy	best	0.745455	0.816092	0.953488	0.836735

Naive Bayes (NB)

precision	accuracy	recall	f1
0.8	0.678161	0.465116	0.588235

Support Vector Machine (SVP)

gamma	kernel	precision	accuracy	recall	f1
scale	rbf	0.717391	0.735632	0.767442	0.741573
scale	linear	0.75	0.735632	0.697674	0.722892
auto	linear	0.75	0.735632	0.697674	0.722892
auto	rbf	0.702128	0.724138	0.767442	0.733333
scale	sigmoid	0.647059	0.678161	0.767442	0.702128
auto	sigmoid	0.647059	0.678161	0.767442	0.702128
auto	poly	0.772727	0.643678	0.395349	0.523077
scale	poly	0.833333	0.597701	0.232558	0.363636

Multi-Layer Perceptron (MLP)

For sake of brevity, only the top 100 results by accuracy are shown.

activation	hidden_layer_sizes	learning_rate	max_iter	solver	precision	accuracy	recall	f1
logistic	(60, 80, 100)	constant	500000	lbfgs	0.895833	0.942529	1	0.945055
logistic	(40, 80, 100)	adaptive	500000	lbfgs	0.86	0.91954	1	0.924731
tanh	(40, 80, 100)	invscaling	500000	adam	0.86	0.91954	1	0.924731
tanh	(60, 100, 80)	adaptive	500000	lbfgs	0.86	0.91954	1	0.924731
tanh	(100, 60, 20)	constant	500000	adam	0.86	0.91954	1	0.924731
tanh	(100, 80, 80)	constant	500000	adam	0.86	0.91954	1	0.924731
relu	(75, 30)	adaptive	500000	lbfgs	0.86	0.91954	1	0.924731
logistic	(20, 40, 60)	adaptive	500000	lbfgs	0.875	0.91954	0.976744	0.923077
logistic	(40, 60, 80)	adaptive	500000	lbfgs	0.843137	0.908046	1	0.914894
logistic	(80, 40, 20)	constant	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	30	invscaling	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	60	constant	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	85	adaptive	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	(30, 30)	constant	500000	adam	0.843137	0.908046	1	0.914894
tanh	(45, 45)	adaptive	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	(60, 60)	invscaling	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	(75, 45)	invscaling	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	(75, 75)	adaptive	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	(90, 90)	invscaling	500000	adam	0.843137	0.908046	1	0.914894
tanh	(20, 40, 60)	invscaling	500000	adam	0.843137	0.908046	1	0.914894
tanh	(20, 100, 20)	invscaling	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	(40, 20, 100)	constant	500000	adam	0.843137	0.908046	1	0.914894
tanh	(40, 80, 60)	invscaling	500000	adam	0.843137	0.908046	1	0.914894
tanh	(40, 80, 100)	adaptive	500000	adam	0.843137	0.908046	1	0.914894
tanh	(60, 20, 40)	adaptive	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	(60, 60, 80)	constant	500000	adam	0.843137	0.908046	1	0.914894
tanh	(60, 80, 80)	adaptive	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	(80, 20, 40)	adaptive	500000	lbfgs	0.843137	0.908046	1	0.914894
tanh	(80, 40, 80)	constant	500000	adam	0.843137	0.908046	1	0.914894
tanh	(80, 60, 60)	constant	500000	lbfgs	0.843137	0.908046	1	0.914894
relu	(20, 20, 80)	constant	500000	adam	0.843137	0.908046	1	0.914894
relu	(20, 40, 100)	constant	500000	lbfgs	0.843137	0.908046	1	0.914894
relu	(20, 60, 20)	adaptive	500000	adam	0.843137	0.908046	1	0.914894
relu	(20, 60, 100)	adaptive	500000	adam	0.843137	0.908046	1	0.914894
relu	(20, 100, 20)	constant	500000	adam	0.843137	0.908046	1	0.914894
relu	(20, 100, 40)	adaptive	500000	adam	0.843137	0.908046	1	0.914894
relu	(40, 20, 80)	constant	500000	lbfgs	0.843137	0.908046	1	0.914894
relu	(40, 80, 60)	constant	500000	lbfgs	0.843137	0.908046	1	0.914894
relu	(60, 20, 100)	constant	500000	adam	0.843137	0.908046	1	0.914894
relu	(80, 20, 60)	constant	500000	lbfgs	0.843137	0.908046	1	0.914894
relu	(80, 60, 20)	adaptive	500000	adam	0.843137	0.908046	1	0.914894
relu	(100, 20, 60)	invscaling	500000	adam	0.843137	0.908046	1	0.914894
logistic	(20, 60, 80)	invscaling	500000	lbfgs	0.857143	0.908046	0.976744	0.913043
logistic	(60, 20, 20)	adaptive	500000	lbfgs	0.857143	0.908046	0.976744	0.913043
tanh	(15, 45)	constant	500000	lbfgs	0.857143	0.908046	0.976744	0.913043
tanh	(45, 90)	invscaling	500000	lbfgs	0.857143	0.908046	0.976744	0.913043
tanh	(90, 30)	constant	500000	lbfgs	0.857143	0.908046	0.976744	0.913043
tanh	(20, 80, 100)	invscaling	500000	adam	0.857143	0.908046	0.976744	0.913043
tanh	(20, 80, 100)	adaptive	500000	lbfgs	0.857143	0.908046	0.976744	0.913043
tanh	(40, 40, 40)	adaptive	500000	lbfgs	0.857143	0.908046	0.976744	0.913043
tanh	(40, 60, 100)	adaptive	500000	adam	0.857143	0.908046	0.976744	0.913043
tanh	(60, 80, 60)	constant	500000	lbfgs	0.857143	0.908046	0.976744	0.913043
tanh	(100, 40, 60)	invscaling	500000	lbfgs	0.857143	0.908046	0.976744	0.913043
tanh	(100, 80, 100)	adaptive	500000	adam	0.857143	0.908046	0.976744	0.913043
relu	(30, 30)	adaptive	500000	lbfgs	0.857143	0.908046	0.976744	0.913043

activation	hidden_layer_sizes	learning_rate	max_iter	solver	precision	accuracy	recall	f1
relu	(20, 20, 40)	adaptive	500000	lbfgs	0.857143	0.908046	0.976744	0.913043
relu	(20, 40, 40)	adaptive	500000	adam	0.857143	0.908046	0.976744	0.913043
relu	(40, 20, 100)	adaptive	500000	adam	0.857143	0.908046	0.976744	0.913043
relu	(60, 80, 20)	invscaling	500000	lbfgs	0.857143	0.908046	0.976744	0.913043
logistic	(40, 80, 60)	adaptive	500000	lbfgs	0.87234	0.908046	0.953488	0.911111
logistic	35	adaptive	500000	lbfgs	0.826923	0.896552	1	0.905263
logistic	(15, 60)	invscaling	500000	lbfgs	0.826923	0.896552	1	0.905263
logistic	(45, 45)	constant	500000	lbfgs	0.826923	0.896552	1	0.905263
logistic	(20, 20, 60)	adaptive	500000	lbfgs	0.826923	0.896552	1	0.905263
logistic	(60, 60, 80)	adaptive	500000	lbfgs	0.826923	0.896552	1	0.905263
logistic	(80, 40, 100)	invscaling	500000	lbfgs	0.826923	0.896552	1	0.905263
logistic	(100, 100, 100)	constant	500000	lbfgs	0.826923	0.896552	1	0.905263
tanh	60	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(15, 15)	invscaling	500000	adam	0.826923	0.896552	1	0.905263
tanh	(15, 45)	adaptive	500000	adam	0.826923	0.896552	1	0.905263
tanh	(30, 30)	invscaling	500000	adam	0.826923	0.896552	1	0.905263
tanh	(30, 60)	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(60, 90)	invscaling	500000	lbfgs	0.826923	0.896552	1	0.905263
tanh	(75, 15)	constant	500000	lbfgs	0.826923	0.896552	1	0.905263
tanh	(75, 45)	constant	500000	lbfgs	0.826923	0.896552	1	0.905263
tanh	(90, 15)	adaptive	500000	adam	0.826923	0.896552	1	0.905263
tanh	(90, 45)	invscaling	500000	lbfgs	0.826923	0.896552	1	0.905263
tanh	(20, 40, 20)	invscaling	500000	adam	0.826923	0.896552	1	0.905263
tanh	(20, 40, 40)	invscaling	500000	adam	0.826923	0.896552	1	0.905263
tanh	(20, 60, 20)	adaptive	500000	adam	0.826923	0.896552	1	0.905263
tanh	(20, 80, 60)	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(20, 80, 80)	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(20, 80, 100)	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(40, 20, 60)	invscaling	500000	lbfgs	0.826923	0.896552	1	0.905263
tanh	(40, 60, 60)	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(40, 60, 60)	invscaling	500000	adam	0.826923	0.896552	1	0.905263
tanh	(40, 80, 20)	adaptive	500000	adam	0.826923	0.896552	1	0.905263
tanh	(40, 100, 60)	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(60, 40, 20)	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(60, 40, 40)	invscaling	500000	adam	0.826923	0.896552	1	0.905263
tanh	(60, 40, 80)	constant	500000	lbfgs	0.826923	0.896552	1	0.905263
tanh	(60, 60, 20)	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(60, 80, 60)	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(60, 80, 80)	invscaling	500000	adam	0.826923	0.896552	1	0.905263
tanh	(60, 100, 20)	invscaling	500000	adam	0.826923	0.896552	1	0.905263
tanh	(60, 100, 40)	adaptive	500000	adam	0.826923	0.896552	1	0.905263
tanh	(60, 100, 60)	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(60, 100, 60)	adaptive	500000	adam	0.826923	0.896552	1	0.905263
tanh	(60, 100, 80)	constant	500000	adam	0.826923	0.896552	1	0.905263
tanh	(80, 40, 40)	constant	500000	adam	0.826923	0.896552	1	0.905263

Random Forest (RF)

criterion	class_weight	max_features	precision	accuracy	recall	f1
gini	balanced	sqrt	0.836735	0.885057	0.953488	0.891304
entropy	balanced	sqrt	0.807692	0.873563	0.976744	0.884211
gini	balanced_subsample	sqrt	0.807692	0.873563	0.976744	0.884211
entropy	balanced_subsample	sqrt	0.807692	0.873563	0.976744	0.884211

criterion	class_weight	max_features	precision	accuracy	recall	f1
gini	balanced	log2	0.82	0.873563	0.953488	0.88172
entropy	balanced	log2	0.82	0.873563	0.953488	0.88172
gini	balanced_subsample	log2	0.803922	0.862069	0.953488	0.87234
entropy	balanced_subsample	log2	0.803922	0.862069	0.953488	0.87234

Evaluation

To evaluate the performance of each selected classifier, each model has trained 100 times by repeating a 5-fold cross validation procedure 20 times. A graphical and statistical analysis to compare the distribution of performance metrics (precision, recall and F1 score) follows.

Numeric tables, statistical analysis conclusions and the boxplot diagram shown in this section are obtained by running the script:

```
./evaluate_classifiers.py
```

Output Distributions

A boxplot chart to show the distribution of each of precision, recall, and F1 score for all classifiers (including the biased classifier) is shown in figure 1. Table 2 is a numeric table summing up mean and standard deviation of each metric.

Table 12: Mean and standard deviation for each classifier for 20-times cross validation.

Classifier	**Metric**	**Mean**	**Std. dev.**
BiasedClassifier	f1	0.666238	0.00511791
BiasedClassifier	precision	0.49954	0.00575757
BiasedClassifier	recall	1	0
DecisionTreeClassifier	f1	0.888104	0.0302085
DecisionTreeClassifier	precision	0.832669	0.0425814
DecisionTreeClassifier	recall	0.953261	0.0358021
GaussianNB	f1	0.54952	0.0912106
GaussianNB	precision	0.820866	0.066323
GaussianNB	recall	0.418885	0.0950382
MLPClassifier	f1	0.884807	0.0348063
MLPClassifier	precision	0.836507	0.0489391
MLPClassifier	recall	0.941823	0.0454851
RandomForestClassifier	f1	0.91079	0.0301481
RandomForestClassifier	precision	0.870685	0.0427383
RandomForestClassifier	recall	0.956665	0.0381288
SVC	f1	0.752656	0.0537908
SVC	precision	0.7557	0.0515768
SVC	recall	0.754709	0.0819603

Comparison and Significance

Given the distribution of metrics presented in the previous section, I perform a statistical analysis using the Wilixcon paired test to determine for each pair of classifiers which one performs better according to each performance metric. When the *p-value* is too high, *ANOVA* power analysis (corrected by $alpha = = 0.05$) is performed to determine if the metrics are equally distributed or if the statistical test is inconclusive.

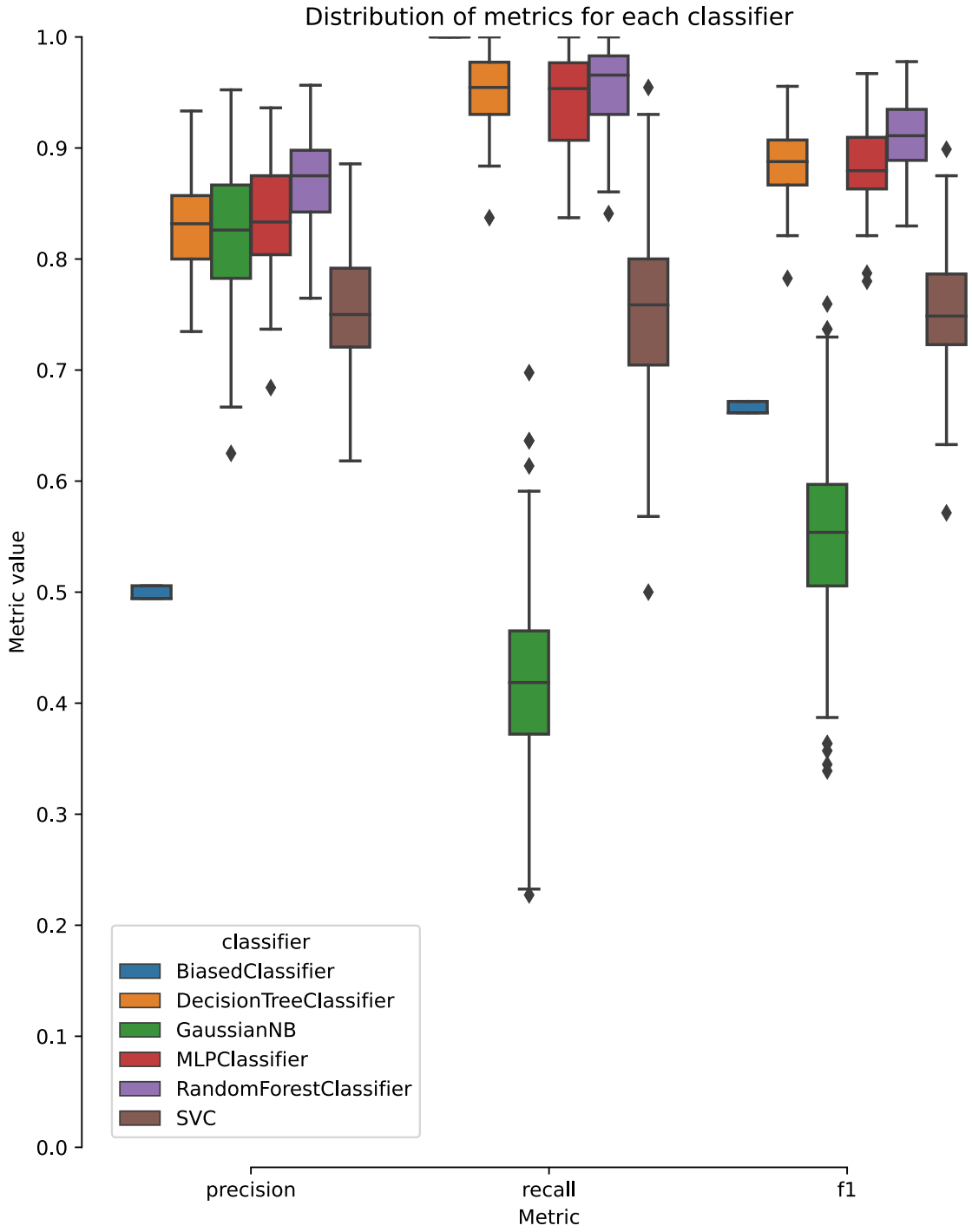

Figure 1: Precision, Recall and F1 score distribution for each classifier for 20-times cross validation.

F1 Values

- Mean *F1* for *DT*: 0.8881, mean *F1* for *NB*: 0.5495 ⇒ *DT* is better than *NB* (*p-value* = 0.0)
- Mean *F1* for *DT*: 0.8881, mean *F1* for *MLP*: 0.8848 ⇒ statistical test inconclusive (*p-value* = 0.4711, *5% corrected ANOVA power* = 0.2987)
- Mean *F1* for *DT*: 0.8881, mean *F1* for *RF*: 0.9108 ⇒ *RF* is better than *DT* (*p-value* = 0.0)
- Mean *F1* for *DT*: 0.8881, mean *F1* for *SVP*: 0.7527 ⇒ *DT* is better than *SVP* (*p-value* = 0.0)
- Mean *F1* for *NB*: 0.5495, mean *F1* for *MLP*: 0.8848 ⇒ *MLP* is better than *NB* (*p-value* = 0.0)
- Mean *F1* for *NB*: 0.5495, mean *F1* for *RF*: 0.9108 ⇒ *RF* is better than *NB* (*p-value* = 0.0)
- Mean *F1* for *NB*: 0.5495, mean *F1* for *SVP*: 0.7527 ⇒ *SVP* is better than *NB* (*p-value* = 0.0)
- Mean *F1* for *MLP*: 0.8848, mean *F1* for *RF*: 0.9108 ⇒ *RF* is better than *MLP* (*p-value* = 0.0)
- Mean *F1* for *MLP*: 0.8848, mean *F1* for *SVP*: 0.7527 ⇒ *MLP* is better than *SVP* (*p-value* = 0.0)
- Mean *F1* for *RF*: 0.9108, mean *F1* for *SVP*: 0.7527 ⇒ *RF* is better than *SVP* (*p-value* = 0.0)
- Mean *F1* for *Biased*: 0.6662, mean *F1* for *DT*: 0.8881 ⇒ *DT* is better than *Biased* (*p-value* = 0.0)
- Mean *F1* for *Biased*: 0.6662, mean *F1* for *NB*: 0.5495 ⇒ *Biased* is better than *NB* (*p-value* = 0.0)
- Mean *F1* for *Biased*: 0.6662, mean *F1* for *MLP*: 0.8848 ⇒ *MLP* is better than *Biased* (*p-value* = 0.0)
- Mean *F1* for *Biased*: 0.6662, mean *F1* for *RF*: 0.9108 ⇒ *RF* is better than *Biased* (*p-value* = 0.0)
- Mean *F1* for *Biased*: 0.6662, mean *F1* for *SVP*: 0.7527 ⇒ *SVP* is better than *Biased* (*p-value* = 0.0)

Precision

- Mean *precision* for *DT*: 0.8327, mean *precision* for *NB*: 0.8209 ⇒ *DT* is as effective as *NB* (*p-value* = 0.0893, *5% corrected ANOVA power* = 0.8498)
- Mean *precision* for *DT*: 0.8327, mean *precision* for *MLP*: 0.8365 ⇒ statistical test inconclusive (*p-value* = 0.4012, *5% corrected ANOVA power* = 0.2196)
- Mean *precision* for *DT*: 0.8327, mean *precision* for *RF*: 0.8707 ⇒ *RF* is better than *DT* (*p-value* = 0.0)
- Mean *precision* for *DT*: 0.8327, mean *precision* for *SVP*: 0.7557 ⇒ *DT* is better than *SVP* (*p-value* = 0.0)
- Mean *precision* for *NB*: 0.8209, mean *precision* for *MLP*: 0.8365 ⇒ *MLP* is better than *NB* (*p-value* = 0.0348)
- Mean *precision* for *NB*: 0.8209, mean *precision* for *RF*: 0.8707 ⇒ *RF* is better than *NB* (*p-value* = 0.0)
- Mean *precision* for *NB*: 0.8209, mean *precision* for *SVP*: 0.7557 ⇒ *NB* is better than *SVP* (*p-value* = 0.0)
- Mean *precision* for *MLP*: 0.8365, mean *precision* for *RF*: 0.8707 ⇒ *RF* is better than *MLP* (*p-value* = 0.0)
- Mean *precision* for *MLP*: 0.8365, mean *precision* for *SVP*: 0.7557 ⇒ *MLP* is better than *SVP* (*p-value* = 0.0)
- Mean *precision* for *RF*: 0.8707, mean *precision* for *SVP*: 0.7557 ⇒ *RF* is better than *SVP* (*p-value* = 0.0)
- Mean *precision* for *Biased*: 0.4995, mean *precision* for *DT*: 0.8327 ⇒ *DT* is better than *Biased* (*p-value* = 0.0)
- Mean *precision* for *Biased*: 0.4995, mean *precision* for *NB*: 0.8209 ⇒ *NB* is better than *Biased* (*p-value* = 0.0)
- Mean *precision* for *Biased*: 0.4995, mean *precision* for *MLP*: 0.8365 ⇒ *MLP* is better than *Biased* (*p-value* = 0.0)
- Mean *precision* for *Biased*: 0.4995, mean *precision* for *RF*: 0.8707 ⇒ *RF* is better than *Biased* (*p-value* = 0.0)
- Mean *precision* for *Biased*: 0.4995, mean *precision* for *SVP*: 0.7557 ⇒ *SVP* is better than *Biased* (*p-value* = 0.0)

Recall

- Mean *recall* for *DT*: 0.9533, mean *recall* for *NB*: 0.4189 ⇒ *DT* is better than *NB* (*p-value* = 0.0)
- Mean *recall* for *DT*: 0.9533, mean *recall* for *MLP*: 0.9418 ⇒ *DT* is better than *MLP* (*p-value* = 0.0118)
- Mean *recall* for *DT*: 0.9533, mean *recall* for *RF*: 0.9567 ⇒ statistical test inconclusive (*p-value* = 0.3276, *5% corrected ANOVA power* = 0.2558)
- Mean *recall* for *DT*: 0.9533, mean *recall* for *SVP*: 0.7547 ⇒ *DT* is better than *SVP* (*p-value* = 0.0)
- Mean *recall* for *NB*: 0.4189, mean *recall* for *MLP*: 0.9418 ⇒ *MLP* is better than *NB* (*p-value* = 0.0)
- Mean *recall* for *NB*: 0.4189, mean *recall* for *RF*: 0.9567 ⇒ *RF* is better than *NB* (*p-value* = 0.0)
- Mean *recall* for *NB*: 0.4189, mean *recall* for *SVP*: 0.7547 ⇒ *SVP* is better than *NB* (*p-value* = 0.0)
- Mean *recall* for *MLP*: 0.9418, mean *recall* for *RF*: 0.9567 ⇒ *RF* is better than *MLP* (*p-value* = 0.0001)
- Mean *recall* for *MLP*: 0.9418, mean *recall* for *SVP*: 0.7547 ⇒ *MLP* is better than *SVP* (*p-value* = 0.0)
- Mean *recall* for *RF*: 0.9567, mean *recall* for *SVP*: 0.7547 ⇒ *RF* is better than *SVP* (*p-value* = 0.0)
- Mean *recall* for *Biased*: 1.0, mean *recall* for *DT*: 0.9533 ⇒ *Biased* is better than *DT* (*p-value* = 0.0)
- Mean *recall* for *Biased*: 1.0, mean *recall* for *NB*: 0.4189 ⇒ *Biased* is better than *NB* (*p-value* = 0.0)
- Mean *recall* for *Biased*: 1.0, mean *recall* for *MLP*: 0.9418 ⇒ *Biased* is better than *MLP* (*p-value* = 0.0)

- Mean *recall* for *Biased*: 1.0, mean *recall* for RF: 0.9567 $\Rightarrow$ *Biased* is better than RF ($p\text{-}value = 0.0$)
- Mean *recall* for *Biased*: 1.0, mean *recall* for SVP: 0.7547 $\Rightarrow$ *Biased* is better than SVP ($p\text{-}value = 0.0$)

Practical Usefulness

The evaluation shows that all classifiers but the Naive Bayes classifier outperform the biased classifier in overall performance (represented by the F1 score). Given the evaluation was not performed on a dedicated test set, all classifiers may be subjected to training bias, which might yield better metrics than a similar evaluation performed on completely new data. However, given the evaluation sample size (i.e. the number of training runs) this effect is minimal.

Given this premise, I can say with reasonable confidence the DT, MLP, RP and SVP classifiers would be useful in predicting potential bugs in the Google JSComp project, i.e. the source of the used dataset. Given the literature presented during lecture, it is not certain if the same trained classifier would yield acceptable results for source code from another project. This is because differences in coding conventions, the bug tracking process, or simply the definition of what constitutes a bug may vary from project to project. A solution to this problem might be to simply train the classifiers on a dataset coming from the project where bug prediction is needed.