Performance Analysis of Machine Learning Frameworks for Software Defect Prediction

Nithya Manoj
181IT131
Information Technology
National Institute of Technology,Karnataka
Surathkal,India 575025
nithyamanoj.ms@gmail.com

Krishna Poojitha Vantakula
181IT223
Information Technology
National Institute of Technology,Karnataka
Surathkal,India 575025
krishnapoojitha2001@gmail.com

Hima Sajeev
181IT254
Information Technology
National Institute of Technology,Karnataka
Surathkal,India 575025
himasajeev0801@gmail.com

Abstract - **The quality of data is generally the most important factor to determine the performance of a classification model. Class imbalance learning refers to learning from data sets that contain significant imbalance among or within classes. The class imbalance of defect datasets will severely affect the prediction performance, especially for the extreme imbalance data classification. Imbalance is associated with the situation in which some classes of data are highly under-represented compared to other classes. By convention, we call the classes having more values the majority classes, and the ones having fewer values the minority classes. Misclassifying an example from the minority class is generally costly.**

The influence that data imbalance has on predictive performance varies from one classifier to another. Hence we intend to perform predictions, evaluate and analyse the stated defects using various machine learning frameworks.

Index Terms - **classification model, class imbalance, defect, prediction, machine learning frameworks**

I. INTRODUCTION

Software Testing is a process of analyzing a software item to detect the differences between existing and required conditions and to evaluate the features of the software item (definition according to ANSI/IEEE 1059 standard). As Agile Software Development became more popular, increased project success and stakeholder satisfaction, along with productivity and quality improvements, have been gained by organizations adopting agile. For testers, the transition to agile is not as easy. There is blissful simplicity in breaking down the work into small, manageable units. Even when a team is only working on one or two features at once, testing is not that simple. There are many tests to run for each of the new features to cover all the scenarios. In the Testing Phase of SDLC, Software Defect Prediction (SDP) is one of the most assisting activities . SDP identifies the modules that are defect prone, which require extensive testing. In this way, the testing resources can be used efficiently. Though SDP is very helpful in testing, it's not easy to predict the defective modules. The smooth performance as well as use of the Defect Prediction models are hindered by various issues.

We worked and analysed a few Machine Learning frameworks combining 4 classification models, data resampling techniques and a feature selection technique, considering datasets containing data defects which in turn cause software defects.

The inherent complex structures and the imbalanced class distribution in the software defect data make it challenging to obtain suitable data features and learn an effective defect prediction model. These challenges can be overcome by the implementation of different combinations of the mentioned algorithms.

On performing several combinations of the standard classification algorithms with resampling and feature selection techniques, the performance of models with the complex and imbalanced dataset improved to a great extent in terms of the metrics used as compared with the standard implementation of models.

II. LITERATURE SURVEY

Software defect prediction aims to predict whether a software module is defect-prone by constructing prediction models. The performance of such SDP models is susceptible to the high dimensionality of the datasets that may include irrelevant and redundant features. Feature selection is applied to alleviate this issue.Menzies et al.[1] found that Naive Bayes classifier with Information Gain based feature selection can get good performances over 10 projects from the NASA dataset. Shivaji et al.[2,3] studied the performance of filter-based and wrapper-based feature selection methods for bug prediction. Their experiments showed that feature selection can improve the defect prediction performance even remaining 10% of the original features. Wold et al.[4] investigated four filter-based feature selection methods on a large telecommunication system and found that the Kolmogorov–Smirnov method achieved the best performance. Catal and Diri [5] conducted an empirical study to investigate the impact of the dataset size, the types of feature sets and the feature selection methods on defect prediction. To study the impact of feature selection methods, they first utilized a Correlation-based Feature Selection (CFS) method to obtain the relevant features before training the classification models. The experiments on five projects from NASA dataset showed that the random forest classifier with CFS performed well on large project datasets and the Naive Bayes classifier with CFS worked well on small projects datasets.

Lessmann et al.[6] conducted an empirical study to investigate the effectiveness of 21 classifiers on NASA dataset. The results showed that the performances of most classifiers have no significant differences. They suggested that some additional factors, such as the computational overhead and simplicity, should be considered when selecting a proper classifier for defect prediction. Ghotra et al.[7] expanded Lessmann's experiment by applying 31 classifiers to two versions of the NASA dataset and PROMISE dataset. The results showed that these classifiers achieved similar results on the noisy NASA dataset but different performance on the clean NASA and the PROMISE datasets.

Mesquita et al.[8] proposed a method based on ELM with a reject option (i.e., IrejoELM) for defect prediction. The results were good because they abandoned the modules that have the contradictory decisions for two designed classifiers. However, in practice, such modules should be considered.

The drawbacks we found from the literature survey include the inability to identify a generalized subset of attributes which act as a substantial factor for a module to be incorrect or non faulty and a great deal of inconsistency in choosing the performance reporting measures. The class imbalance problem of the available dataset is also one of the hindering factors. Another issue with this upcoming field is that different scholars and researchers have used different techniques on different data sets. But, there has been no standard framework or procedure to apply a software defect prediction process on a local or cross company project.

III. PROPOSED METHODOLOGY

From surveying existing models and analyzing datasets, three major challenges were identified in software defect prediction problems which were resolved in the flow shown in Figure 1.

1. Class imbalance problem: All the available datasets are heavily skewed, containing way less defective data than clean. To take care of the class imbalance problem, we decided to make use of various resampling techniques.

2.Similar features problem: Features in Software defect prediction models tend to be dependent on each other and often provide no real value to the training algorithm.
In addition to the widely used PCA feature selection technique, 2 more techniques - KPCA and a statistical Information gain technique was employed.

3.Choosing predictive model: Various common machine learning algorithms were used in predicting and comparing to one another based on f1-score and area under the ROC-AUC curve.

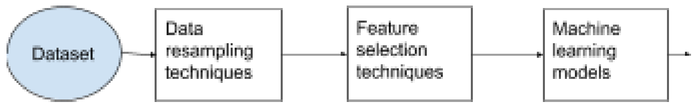


Figure 1: Flow of the dataset in the process

A. Work Done

Data resampling techniques

1. Random UnderSampler: This technique involves randomly duplicating examples from the minority class and adding them to the training dataset.

2. Random Oversampler: It involves randomly selecting examples from the majority class to delete from the training dataset.

3. SMOTE Oversampler: This is a statistical technique for increasing the number of cases in your dataset in a balanced way. The module works by generating new instances from existing minority cases that you supply as input.

Various combinations of these sampling techniques were used to train and were compared against each other.

Feature selection techniques

1. Principal Component Analysis: The main idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, while retaining the variation present in the dataset, up to the maximum extent.

2. Kernel Principal Component Analysis: It is an extension of principal component analysis (PCA)

using techniques of kernel methods. Using a kernel, the originally linear operations of PCA are performed in a reproducing kernel Hilbert space.

3. Information gain: It is a filter based feature ranking method. Information gain is basically the change in entropy before and after transforming the dataset in some way. In this method, for every feature information gain is calculated, and only the features that maximise information gain are used.

Classification Models

1. Random Forest: It is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

2. Neural Networks: It consists of units (neurons), arranged in layers, which convert an input vector into some output. Each unit takes an input, applies a (often nonlinear) function to it and then passes the output on to the next layer.

3. Support Vector Machine: The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N - the number of features) that distinctly classifies the data points that have the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

4. XGBoost: It is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework.

Metrics

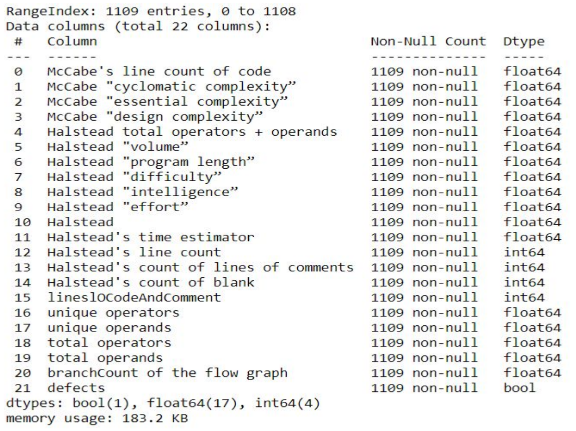
1. F1-score: It is the harmonic mean of Precision and Recall and gives a better measure of the incorrectly classified cases than the Accuracy Metric. It is a better metric when there are imbalanced classes.

2. ROC-AUC score: This is the performance measurement for classification problems at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much a model is capable of distinguishing between classes.

B. Dataset

We implemented the models using the following datasets :

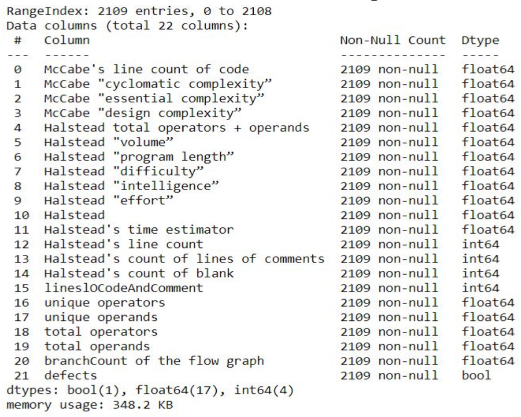
- pc1.csv - Contains 1109 data points.



```
RangeIndex: 1109 entries, 0 to 1108
Data columns (total 22 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   McCabe's line count of code         1109 non-null   float64
 1   McCabe "cyclomatic complexity"      1109 non-null   float64
 2   McCabe "essential complexity"       1109 non-null   float64
 3   McCabe "design complexity"          1109 non-null   float64
 4   Halstead total operators + operands 1109 non-null   float64
 5   Halstead "volume"                   1109 non-null   float64
 6   Halstead "program length"           1109 non-null   float64
 7   Halstead "difficulty"               1109 non-null   float64
 8   Halstead "intelligence"             1109 non-null   float64
 9   Halstead "effort"                   1109 non-null   float64
 10  Halstead                            1109 non-null   float64
 11  Halstead's time estimator           1109 non-null   float64
 12  Halstead's line count               1109 non-null   int64
 13  Halstead's count of lines of comments 1109 non-null int64
 14  Halstead's count of blank           1109 non-null   int64
 15  lineslOCodeAndComment               1109 non-null   int64
 16  unique operators                    1109 non-null   float64
 17  unique operands                     1109 non-null   float64
 18  total operators                     1109 non-null   float64
 19  total operands                      1109 non-null   float64
 20  branchCount of the flow graph       1109 non-null   float64
 21  defects                             1109 non-null   bool
dtypes: bool(1), float64(17), int64(4)
memory usage: 183.2 KB
```

Figure 2: Description of the dataset pc1.csv

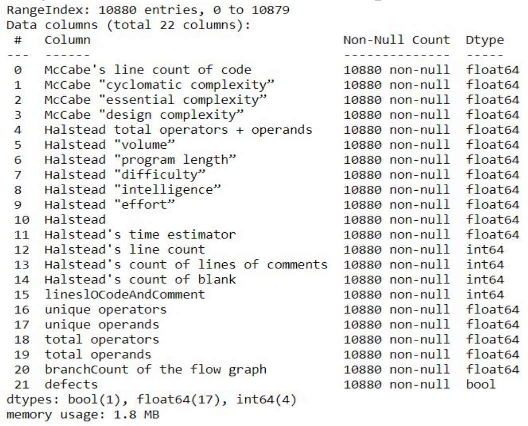
- kc1.csv - Contains 2109 data points.



```
RangeIndex: 2109 entries, 0 to 2108
Data columns (total 22 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   McCabe's line count of code         2109 non-null   float64
 1   McCabe "cyclomatic complexity"      2109 non-null   float64
 2   McCabe "essential complexity"       2109 non-null   float64
 3   McCabe "design complexity"          2109 non-null   float64
 4   Halstead total operators + operands 2109 non-null   float64
 5   Halstead "volume"                   2109 non-null   float64
 6   Halstead "program length"           2109 non-null   float64
 7   Halstead "difficulty"               2109 non-null   float64
 8   Halstead "intelligence"             2109 non-null   float64
 9   Halstead "effort"                   2109 non-null   float64
 10  Halstead                            2109 non-null   float64
 11  Halstead's time estimator           2109 non-null   float64
 12  Halstead's line count               2109 non-null   int64
 13  Halstead's count of lines of comments 2109 non-null int64
 14  Halstead's count of blank           2109 non-null   int64
 15  lineslOCodeAndComment               2109 non-null   int64
 16  unique operators                    2109 non-null   float64
 17  unique operands                     2109 non-null   float64
 18  total operators                     2109 non-null   float64
 19  total operands                      2109 non-null   float64
 20  branchCount of the flow graph       2109 non-null   float64
 21  defects                             2109 non-null   bool
dtypes: bool(1), float64(17), int64(4)
memory usage: 348.2 KB
```

Figure 3: Description of the dataset kc1.csv

- jm1.csv - Contains 10880 data points.



```
RangeIndex: 10880 entries, 0 to 10879
Data columns (total 22 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   McCabe's line count of code         10880 non-null  float64
 1   McCabe "cyclomatic complexity"      10880 non-null  float64
 2   McCabe "essential complexity"       10880 non-null  float64
 3   McCabe "design complexity"          10880 non-null  float64
 4   Halstead total operators + operands 10880 non-null  float64
 5   Halstead "volume"                   10880 non-null  float64
 6   Halstead "program length"           10880 non-null  float64
 7   Halstead "difficulty"               10880 non-null  float64
 8   Halstead "intelligence"             10880 non-null  float64
 9   Halstead "effort"                   10880 non-null  float64
 10  Halstead                            10880 non-null  float64
 11  Halstead's time estimator           10880 non-null  float64
 12  Halstead's line count               10880 non-null  int64
 13  Halstead's count of lines of comments 10880 non-null int64
 14  Halstead's count of blank           10880 non-null  int64
 15  lineslOCodeAndComment               10880 non-null  int64
 16  unique operators                    10880 non-null  float64
 17  unique operands                     10880 non-null  float64
 18  total operators                     10880 non-null  float64
 19  total operands                      10880 non-null  float64
 20  branchCount of the flow graph       10880 non-null  float64
 21  defects                             10880 non-null  bool
dtypes: bool(1), float64(17), int64(4)
memory usage: 1.8 MB
```

Figure 4: Description of the dataset jm1.csv

C. Implementation

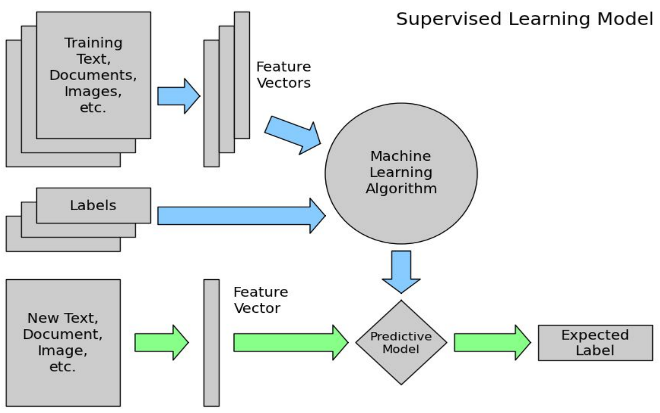

Supervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

Figure 5: Implementation flow diagram

The steps for implementation of the proposed machine learning frameworks following the Figure 5 are:

1. The structured dataset is read from the .csv file and converted to dataframe using the pandas library.

2. The target values of the dataset are converted into suitable format for training and predicting using LabelEncoder from sklearn.preprocessing.

3. Dataset is split into training and testing sets of 80% and 20% respectively, along with 5-fold validation.

4. Resampling techniques are then applied to the training dataset in order to train the model with sufficient and appropriate data of both the classes.

5. Feature scaling is performed to the training and testing data using StandardScaler from sklearn.preprocessing to avoid the superiority of few features with greater magnitudes compared to others with lower magnitudes.

6. The classification model used is imported from sklearn library, which is then fed with the training dataset.

7. Predictions of the target values are made for the testing dataset from the trained classification model.

8. Statistics of the correlation among target values predicted and true values are displayed using confusion_matrix from sklearn.metrics.

9. The metrics used for comparing performance of models, f1_score and roc_auc_score are imported from sklearn.metrics along with it the ROC curve is plotted using matplotlib.pyplot.

IV. Results

Initially the machine learning models were implemented with different resampling methods without using any feature selection to check the performance of each sampling technique. The dataset jm1.csv is used to obtain the results shown in Table1 for F1-score and Table2 for ROC-AUC score metrics.

Table 1: F1-score comparison of different models without feature selection of jm1.csv

Implementation	Random Forest	Neural Network	SVM	XGBoost
Without any sampling	0.25954	0.156448	0.175097	0.271604
Undersampling (majority)	0.408152	0.383759	0.40153	0.356107
Undersampling (0.5)	0.405363	0.384505	0.310645	0.368781
Oversampling-random(minority)	0.4025	0.392035	0.38974	0.350799
Oversampling-SMOTE(minority)	0.405816	0.383611	0.40282	0.361963
Oversampling (0.5)	0.383358	0.38678	0.398492	0.379121
Undersampling (0.3)+SMOTE Oversampling(0.8)	0.435669	0.397422	0.433939	0.391223

Table 2: ROC-AUC score comparison of different models without feature selection of jm1.csv

Implementation	Random Forest	Neural Network	SVM	XGBoost
Without data sampling	0.569295	0.536275	0.542783	0.571906
Undersampling (majority)	0.659724	0.660540	0.66398	0.656269
Undersampling (0.5)	0.645818	0.621448	0.587592	0.643200
Oversampling-random (minority)	0.645934	0.680283	0.665424	0.661758
Oversampling -SMOTE(minority)	0.641270	0.661422	0.660839	0.609667
Oversampling (0.5)	0.620566	0.633636	0.586465	0.618830
Undersampling (0.3)+SMOTE Oversampling(0.8)	0.653217	0.633490	0.654178	0.619758

The remaining implementation in then made using different feature selection techniques on all the machine learning models for the following datasets:

- pc1.csv: false entries: 1032, true entries: 77
- kc1.csv: false entries: 1783, true entries: 326
- jm1.csv: false entries: 8777, true entries: 2103

The results obtained from above implementations are shown in Table3 for F1-score and Table4 for ROC-AUC score metrics.

Table 3: F1-score comparison

Dataset	Feature Selection	Random Forest	Neural Network	SVM	XGBoost
pc1.csv	**Normal**	0.21056	0.000000	0.181818	0.45454
	PCA	0.366666	0.363636	0.433333	0.448275
	KPCA	0.431372	0.355769	0.310344	0.44897
	Information gain	0.379310	0.338983	0.360000	0.379310
kc1.csv	**Normal**	0.309278	0.2790697	0.197530	0.26000
	PCA	0.3953488	0.343750	0.3352601	0.41212121
	KPCA	0.3859649	0.35576923	0.3483870	0.43529411
	Information gain	0.40268456	0.38613861	0.35632183	0.41428571
jm1.csv	**Normal**	0.259542	0.156448	0.175097	0.271604
	PCA	0.422989	0.413580	0.431845	0.432989
	KPCA	0.442812	0.418743	0.418743	0.418743
	Information gain	0.4271099	0.437311	0.437118	0.406732

Table 4: ROC-AUC score comparison

Dataset	Feature Selection	Random Forest	Neural Network	SVM	XGBoost
pc1.csv	**Normal**	0.560073	0.5	0.534523	0.68273
	PCA	0.763652	0.7421116	0.831007	0.835861
	KPCA	0.785497	0.623640	0.7011529	0.790351
	Information gain	0.768507	0.732402	0.720570	0.7344827
kc1.csv	**Normal**	0.590282	0.579874	0.552305	0.568707
	PCA	0.648643	0.611651	0.603912	0.658558
	KPCA	0.641396	0.623640	0.612082	0.677464
	Information gain	0.64656	0.649464	0.619821	0.650654
jm1.csv	**Normal**	0.569295	0.536275	0.542783	0.571906
	PCA	0.642071	0.646398	0.653051	0.660718
	KPCA	0.667035	0.6514228	0.6514228	0.651422
	Information gain	0.647947	0.665341	0.655868	0.633755

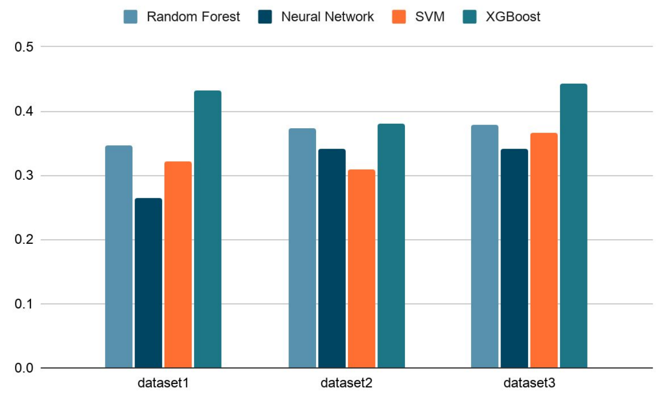


Figure 6: Average F1-score for classification models

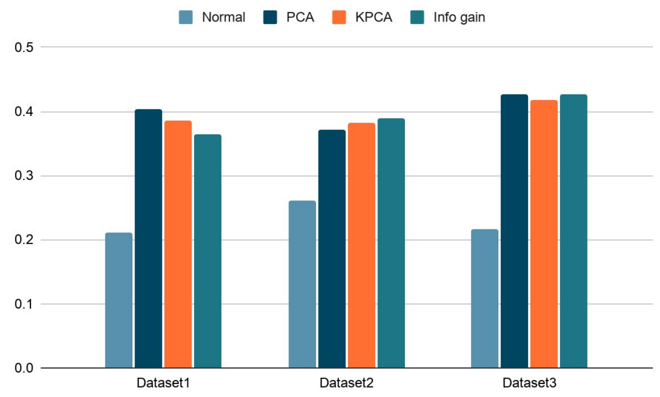


Figure 7: Average F1- scores for feature selection techniques

V. Conclusion

Software Defect Prediction is one of the most essential activities of the Software Development Life Cycle. It identifies the modules that are defect prone and require extensive testing. Data with defects turns out to be a software defect. The traditional models turned out to obtain lower F1-score values as compared to other models with mentioned combinations of techniques.

After validating on a combination of data resampling techniques, a pipeline of RandomUnderSampler(0.3) and SMOTE oversampler(0.8) was chosen. This increases the average F1-score of the models by **192%**.

On comparing the ML algorithms, though XGBoost fared slightly better on average,performance of models was found to be heavily dependent on the dataset chosen and other external factors.

All the employed feature selection techniques produced heavily improved ,comparable performance of models both in terms of f1 score and roc. But other factors like computational needs must be taken into account before choosing a model.

Since information gain is light compared to others, we propose that **information gain** to be used as a feature selection technique. It further **outperforms the classic models** on an average by about **71%** considering the **F1-score** since it is the appropriate metric in case of imbalanced datasets.

References

[1] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. (TSE) 33 (1) (2007) 2–13.

[2] S. Shivaji, E.J. Whitehead, R. Akella, S. Kim, Reducing features to improve code change-based bug prediction, IEEE Trans. Softw. Eng. (TSE) 39 (4) (2013) 552–569.

[3] S. Shivaji, J.E.J. Whitehead, R. Akella, S. Kim, Reducing features to improve bug prediction, in: Proceedings of the 24th International Conference on Automated Software Engineering (ASE), IEEE Computer Society, 2009, pp. 600–604.

[4] S. Wold, K. Esbensen, P. Geladi, Principal component analysis, Chemom. Intell. Lab. Syst. 2 (1–3) (1987) 37–52.

[5] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, Inf. Sci. (Ny) 179 (8) (2009) 1040–1058.

[6] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Trans. Softw. Eng. (TSE) 34 (4) (2008) 485–496.

[7] B. Ghotra, S. McIntosh, A.E. Hassan, A large-scale study of the impact of feature selection techniques on defect classification models, in: Proceedings of the 14th International Conference on Mining Software Repositories (MSR), IEEE, 2017, pp. 146–157.

[8] D.P. Mesquita, L.S. Rocha, J.P.P. Gomes, A.R.R. Neto, Classification with reject option for software defect prediction, Appl. Softw. Comput. 49 (2016) 1085–1093.