International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 6, Issue 5, September- October 2017 ISSN 2278-6856

Empirical Analysis of Open Source projects using Feature Selection and Filtering Techniques

Prabujeet Kaur[1], Dharmendra Lal Gupta[2]

[1]M.tech Student, Department of Computer Science and Engg, KNIT, Dr A.P.J. Abdul Kalam Technical University, Sultanpur, UP, India
[2]Associate Professor, Department of Computer Science and Engg, KNIT, Dr A.P.J. Abdul Kalam Technical University, Sultanpur, UP, India

Abstract: ***To achieve software quality for the large systems results very costly. Developers and testers put a lot of effort to investigate a large number of modules in order to ensure the software quality. This tends to be very time consuming process. The machine learning techniques are used for fault prediction of the modules. However, there are many modules which are very low in priority for quality investigation. In this research, Wrapper subset evaluation method has been used to identify that subset of attributes which are most prominent. 10, 20 and 30% of less complex faulty software modules are filtered out from each attribute. The remaining modules are used to build models against four classifiers: Naïve Bayes, Support Vector Machine, k nearest neighbors and C4.5 decision trees. The results of the classifiers were analyzed and compared against the filtering of less complex instances from LOC and NPM metrics. The classifiers based on wrapper subset evaluation method gave better results than the filtering of LOC and NPM metrics.***

Keywords: Software fault, complexity, CBO, NOA, NMI, DIT, NOC, NAI, NPRIM, NPM, FAN-IN, FAN-OUT

1. INTRODUCTION

Software metrics provide a measurement to evaluate the quality of the software by identifying the faulty parts of systems. When producing high quality software's then it leads to a drastic rise in the software costs. Software engineers can make use of these metrics to analyze the current quality of large software systems for predicting where the quality can be improved. Various static analysis tools are available and data for metrics can also be collected very easily. However, mostly all current software systems are large in size and to ensure their quality requires a lot of development and maintenance effort. A class is made out of methods (public, private, and protected) and data abstractions (primitives, and references). A class size may vary on the basis of the abstraction level, encapsulation, inheritance, and polymorphism purposes. Some classes are large to very large and some classes are small to very small in size and/or complexity. Large classes are usually referred as God classes that can be re-factored either by extracting classes, methods, super-classes, or sub-classes. Small classes are very small and usually have less/ very less complexity. Small classes can be trivial in nature and may require less time to investigate their quality. Software engineers need to allocate their resources efficiently and importantly to those parts of the software which require more efforts.

Advanced tools are required to evaluate the quality of the large systems. Data mining technique is used to classify modules into either faulty or not faulty through metric values [1, 2, 3, 4, 5, 6, 7 and 8]. Software quality evaluated using data mining tools may not give good result if the data is of poor quality due to many reasons like noise [9], class imbalance [10] or redundancies. Therefore, the effect of preprocessing is required before predicting the quality of software modules. Datasets are preprocessed to select the most prominent attributes among all and for this Wrapper subset evaluation method has been used which resulted into different attributes subsets for different open source projects. Further, the datasets are filtered out and kept only larger faulty modules for prediction process. The smaller faulty modules are filtered out from metrics. Three filters viz. 10, 20 and 30% of faulty data are proposed.

The overall structure of the paper is: Section 2 discusses about the related work for fault prediction in software. Section 3 discusses about the research methodology utilized in this research and also give details on data preprocessing, filtering of data, definitions of software metrics used, collection of data and various classification techniques used. Section 4 presents the results analysis of the research on four classifier algorithms. Section 5 shows the comparison between the result of the research obtained and performance of filtering out less complex instances using LOC and NPM metrics. At last Section 6 leads to conclusion and future scope of the research.

2. RELATED WORK

Fault prediction models face many difficulties which incorporate data quality (e.g., noisy data) as well as class imbalance problem [11]. These challenges affect the performance and usage of models in practice. Therefore,

many researchers have introduced different techniques to improve fault prediction models.

Boetticher G [12] has studied the effect of preprocessing on NASA data by removing replicated instances.

Schro¨ter A, Zimmermann T and Zeller A [13] have conducted a study on 52 ECLIPSE plug-ins and selected the training set from the faulty parts only.

Kim S, Zimmermann T, Whitehead E and Zeller A [14], authors have discovered that 10% of modules represent more than73% of faults in seven open-source projects.

Jiang Y, Cukic B and Ma Y [15], authors have tested the effect of log and discretization transformation techniques on ten classifier algorithms yet they could not find any dominant technique. In another study, author has noted a correlation amongst the most critical parts of the code that have been identified using a fault-prediction process and cost required to test those parts [1].

Liebchen and Shepperd [16], they have reported that just 23 out of several hundreds of fault-prediction studies have explicitly considered about the data quality while other faulty models were built without the data cleaning [11].

Gray D, Bowes D, Davey N, Sun Y and Christianson B [17], they have conducted a data preprocessing on NASA metrics and have eliminated about 6 to 90% of the original data by conducting several cleaning processes such as removal of replicated attributes, removal of replicated and inconsistent instances, etc.

Catal C, Alan O and Balkan K [9], they have used metrics thresholds method to find out two types of instances as noisy. Firstly, non-faulty instance is considered as noisy if the majority of the metric values exceeded their corresponding thresholds. Secondly, an instance with faulty class is considered as noisy if all the values of metric are below their corresponding threshold values.

Gao K, Khoshgoftaar TM and Seliya N [18], they have conducted an analysis using many sampling techniques there-after feature selection techniques has been applied for the improvisation of the fault prediction.

Al Dallal [19], he has conducted an empirical study in order to find out the effect cohesion among classes by including or excluding the special methods such as constructors, destructors, and access methods for fault prediction. The results have shown much effect in cohesion measurements and refactoring prediction but not much effect on fault prediction processes.

Shepperd M, Song Q, Sun Z and Mair C [20], authors have applied 18 referential integrity checks for data validity and found large number of errors as well. Data were categorized into two groups. Firstly, the data is said to be problematic if it has impossible values; and secondly, the data is said to be non problematic if it has repeated attributes and does not help in fault prediction.

Recently, Petric´ J, Bowes D, Hall T, Christianson B and Baddoo N [21], they have introduced another data integrity checks on NASA datasets for data cleaning. Authors have added two more integrity checks in addition to the work of [20], but after applying new integrity checks, authors have not done fault prediction. The use of thresholds values for identifying less complex instances can also be related to this work.

Erni and Lewerentz [22], the authors proposed the use of mean and standard deviation in order to find two possible thresholds, the minimum threshold denoted as Tmin and the maximum threshold denoted as Tmax and can be calculated as, Tmin = l – s and Tmax = l ? s, where l being the average of a metric and s the standard deviation. In this research, Tmin are identified so as to eliminate the less complex modules. Thorough and exhaustive quality assurance is very costly and impractical to implement; subsequently, engineers skew their effort toward the key parts of software [23].

In this research, prediction models are built by filtering out less complex faulty instances out of the original data and are cost-effective as well.

3. RESEARCH METHODOLOGY

Data quality is vital to improve the quality of prediction models. An empirical investigation is carried out in order to watch the impact of preprocessing techniques on the performance of fault prediction models.

3.1 Data Sources: This research includes 5 open source projects. They are:

Dataset	Modules	NFP	FP
Eclipse JDT Core www.eclipse.org/jdt/core	997	86%	14%
Equinox framework www.eclipse.org/equinox/	324	60%	40%
Mylyn www.eclipse.org/mylyn/	1862	87%	13%
Eclipse PDE UI www.eclipse.org/pde/pde-ui/	1497	79%	21%
Apache Lucene www.lucene.apache.org	691	91%	9%

The dataset for these projects is publically available at [24].

3.2 Attribute Selection: Before applying preprocessing technique on the data, attribute selection methodology has been applied to identify the most prominent attributes among all. For this, Wrapper subset evaluation technique has been applied with Naïve Bayes classifier at 10 folds cross validation and 0.05 thresholds as its configuration. And Best Fit search evaluator has been used for searching. The metrics obtained are:

3.2.1CBO*:* It is a total number of classes that are coupled to a class through method calls, return types, or exceptions [36].

3.2.2NOA*:* It counts the number of attributes in a class; and for a package it counts the average number of attributes per class of a package [38].

3.2.3NMI: It is the count of the total number methods that are inherited by the child class from the parent class [37].

3.2.4DIT: It is the maximum length from the node of a tree to the root of a tree and is measured as the count of number of ancestor classes [36].

3.2.5NOC: It is a total number of immediate sub-classes of class [36].

3.2.6NAI: It is the count of the total number methods that are inherited by the child class from the parent class [37].

3.2.7NPRIM: It is a total number of methods that are declared within a class [37].

3.2.8NPM*:* It is a total number of methods in a class that are actually declared as public methods [36].

3.2.9FAN-IN*:* It is a count of number of methods or functions that call some other method or function [39].

3.2.10 FAN-OUT*:* It is a count of number of methods or functions that are called by another method or function [39].

3.3 Data Filtering*:* After attaining the attributes, we applied 10%, 20% and 30% filtering on the obtained metrics. 10, 20 and 30% of less complex faulty instances are filtered out, which lead to the formation of 3 new datasets for each corresponding metrics.

3.4 Classification techniques and performance evaluation*:* There are many classifier algorithms available. But the study is conducted on four specific classifier algorithms named Naïve Bayes (NB), Support Vector Machine (SVM), k- nearest neighbors (kNN), and C4.5 decision trees. Weka tool is commonly used for training and testing these classifiers [25].

Naive Bayes (NB) classifier is widely used for fault prediction [26, 27]. It is very simple and uses simple Bayesian network that follows two assumptions. Firstly, the attributes or metrics are independent where as the classes may be faulty or non-faulty and secondly, no hidden or underlying attributes can affect the prediction process [28]. The SVM classifier is a binary algorithm that keeps the margin at its maximum. The separator called hyper-plane is always parallel to the margin planes and is always in midway between the margin planes. Each margin plane passes through point(s) that belong to a particular class and is closest to the margin plane of the other class. The distance between these margin planes is called the margin. One thing to note here is that multiple pairs of margin planes are possible with different margins. But, SVM finds the maximum margin from both the sides of the hyper-plane. The points from each class that pass through the margin planes or that determine the margin planes are termed as the support vectors [35].The nearest neighbors (kNN) algorithm measures the distance or the similarity between the modules using metric values and assign the modules to be either faulty or non faulty as per the dominancy of the closest group of k nodes or objects [29]. The value of K is usually set to be an odd value and in this research, k = 5 has been selected. The selection of the best k is not an objective of this research. The nearest neighbors' algorithm has been used in many previous researches for fault prediction [2, 30 and 31]. C4.5 decision tree classifier uses information based approach viz. information gain to construct decision trees [32]. The tree grows by selecting the metric value with the highest information gain. C4.5 decision tree classifier has been used for in many researches for fault prediction [33 and 34]. All the classifiers use tenfold cross-validation [40].

Performance measures chosen for this research is:

1) *Accuracy*: (TP+TN)/ (TP+FP+TN+FN) can be calculated from the confusion matrix.

4. RESULT ANALYSIS

Below there are measures shown for the accuracy in all the five projects at No filter, 10%, 20% and 30% of filtering out of faulty instances. The result and performance measure is shown below.

Table 1: Accuracy measure for eclipse JDT with no filter

Eclipse JDT for NO Filter				
	NB	SMO	kNN	C4.5
CBO	80.74	79.84	79.24	78.44
NOA	80.74	79.84	79.24	78.44
NMI	80.74	79.84	79.24	78.44
LOC [40]	77.23	77.53	79.23	77.63
NPM [40]	77.23	77.53	79.23	77.63

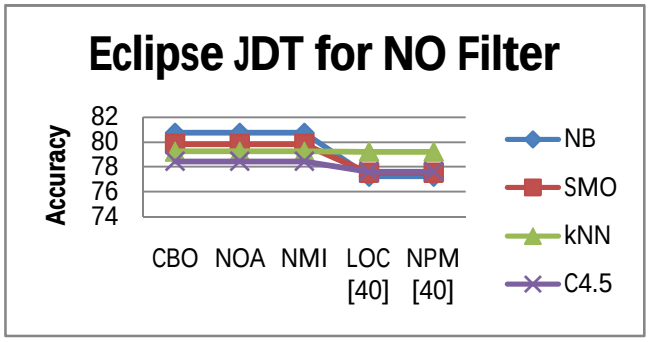

Fig 1: Accuracy graph of Eclipse JDT with No filter

In the above Table 1 and Fig 1, we have shown the accuracy measure for Eclipse JDT with No Filter, where we can see that the metrics named CBO, NOA, and NMI gave the better results than LOC [40] and NPM [40], for all the four classifiers.

Table 2: Accuracy measure for eclipse JDT with 10% filter

Eclipse JDT for 10% Filtering of faulty data				
	NB	SMO	kNN	C4.5
CBO	81.57	81.06	80.66	81.06
NOA	80.86	80.76	80.96	80.76
NMI	80.76	80.66	80.86	80.66
LOC [40]	78.15	76.47	78.48	76.47
NPM [40]	78.59	78.7	78.7	78.7

Table 3: Accuracy measure for eclipse JDT with 20% filter

Eclipse JDT for 20% Filtering of faulty data				
	NB	SMO	kNN	C4.5
CBO	83.37	82.74	82.22	82.74
NOA	82.63	82.53	82.74	82.53
NMI	83.05	83.05	82.85	83.05
LOC [40]	76.19	73.93	76.56	73.93
NPM [40]	77.69	77.94	77.82	77.94

Table 4: Accuracy measure for eclipse JDT with 30% filter

Eclipse JDT for 30% Filtering of faulty data				
	NB	SMO	kNN	C4.5
CBO	85.03	84.71	84.17	84.71
NOA	84.6	84.5	84.6	84.5
NMI	84.6	84.81	84.6	84.81
LOC [40]	73.21	70.77	73.49	68.76
NPM [40]	75.93	76.21	76.21	75.78

Table 5: Accuracy measure for Equinox with No filter

Equinox for NO Filter				
	NB	SMO	kNN	C4.5
DIT	64.81	62.96	59	59.26
NOC	64.81	62.96	59	59.26
NAI	64.81	62.96	59	59.26
NPRIM	64.81	62.96	59	59.26
NPM	64.81	62.96	59	59.26
LOC [40]	58.95	56.48	57.71	59.87
NPM [40]	58.95	56.48	57.71	59.87

Table 6: Accuracy measure for Equinox with 10% filter

Equinox for 10% Filtering of faulty data				
	NB	SMO	kNN	C4.5
DIT	63.98	63.99	64	63.98
NOC	63.66	63.67	64	63.66
NAI	63.34	60.77	61.41	62.06
NPRIM	63.02	61.74	61.74	61.73
NPM	59.81	58.84	60.45	59.16
LOC [40]	58.56	56.85	56.85	56.85
NPM [40]	53.42	50.68	52.05	51.37

Table 7: Accuracy measure for Equinox with 20% filter

Equinox for 20% Filtering of faulty data				
	NB	SMO	kNN	C4.5
DIT	65.43	65.44	65.44	65.44
NOC	66.44	66.44	66.78	66.44
NAI	65.1	62.42	61.75	62.41
NPRIM	65.1	65.44	65.1	65.44
NPM	61.07	59.4	61.07	59.4
LOC [40]	53.28	52.51	51.73	52.51
NPM [40]	49.03	49.03	52.12	51.35

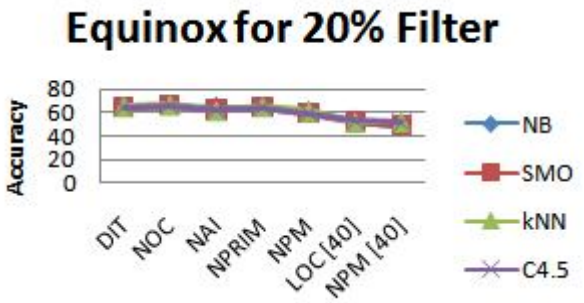

Fig 2: Accuracy graph of Equinox with 20% filter

In Table 7 and Fig 2, we have shown the accuracy measure for Equinox with 20% filtering of less complex faulty instances, where we can see that the metrics named DIT, NOC, NAI, NPRIM and NPM gave the better results than LOC [40] and NPM [40], for all the four classifiers.

Table 8: Accuracy measure for Equinox with 30% filter

Equinox for 30% Filtering of faulty data				
	NB	SMO	kNN	C4.5
DIT	69.82	69.82	69.82	69.82
NOC	69.47	69.47	69.82	69.47
NAI	68.42	66	67.72	67.72
NPRIM	68.77	68.42	68.42	68.42
NPM	65.26	64.91	67.02	64.91
LOC [40]	46.25	45.37	46.7	45.37
NPM [40]	41.85	40.52	42.73	40.52

Table 9: Accuracy measure for Lucene with No filter

Lucene for NO Filter				
	NB	SMO	kNN	C4.5
fanOut	91.02	91.02	90.74	90.74
LOC [40]	89.14	89.43	90.59	86.97
NPM [40]	89.14	89.43	90.59	86.97

Table 10: Accuracy measure for Lucene with 10% filter

Lucene for 10% Filtering of faulty data				
	NB	SMO	kNN	C4.5
fanOut	91.82	91.82	91.53	91.82
LOC [40]	90.19	88.74	90.19	88.74
NPM [40]	90.03	89.87	90.35	89.87

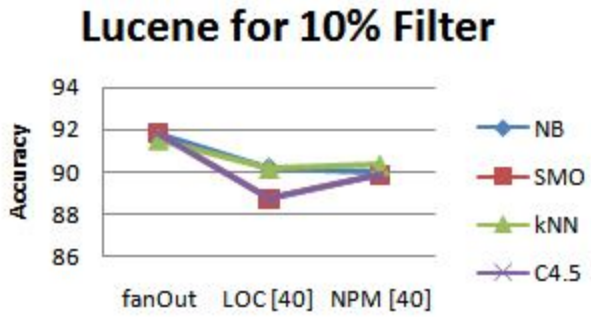

Fig 3: Accuracy graph of Lucene with 10% filter

In Table 10 and Fig 3, we have shown the accuracy measure for Lucene with 10% filtering of less complex faulty instances, where we can see that the metrics named fanOut gave the better results than LOC [40] and NPM [40], for all the four classifiers.

Table 11: Accuracy measure for Lucene with 20% filter

Lucene for 20% Filtering of faulty data				
	NB	SMO	kNN	C4.5
fanOut	92.77	92.77	92.48	92.77
LOC [40]	89.51	87.88	89.51	87.88
NPM [40]	89.51	89.33	89.87	89.33

Table 12: Accuracy measure for Lucene with 30% filter

Lucene for 30% Filtering of faulty data				
	NB	SMO	kNN	C4.5
fanOut	93.6	93.3	93.3	93.3
LOC [40]	88.43	86.77	88.43	86.77
NPM [40]	89.05	88.84	89.46	88.84

Table 13: Accuracy measure for Mylyn with No filter

Mylyn for NO Filter				
	NB	SMO	kNN	C4.5
fanIn	87.11	87.06	86.95	86.84
NAI	87.11	87.06	86.95	86.84
LOC [40]	85.33	85.39	86.73	84.9
NPM[40]	85.33	85.39	86.73	84.9

Table 14: Accuracy measure for Mylyn with 10% filter

Mylyn for 10% Filtering of faulty data				
	NB	SMO	kNN	C4.5
fanIn	88.08	88.03	88.14	88.03
NAI	88.2	88.14	87.98	88.14
LOC [40]	86.09	84.9	86.15	84.9
NPM[40]	86.33	86.27	86.57	86.15

Table 15: Accuracy measure for Mylyn with 20% filter

Mylyn for 20% Filtering of faulty data				
	NB	SMO	kNN	C4.5
fanIn	89.3	89.19	89.3	89.24
NAI	89.41	89.35	89.19	89.35
LOC [40]	85.7	84.43	85.83	84.43
NPM[40]	86.57	86.51	86.71	86.51

Table 16: Accuracy measure for Mylyn with 30% filter

Mylyn for 30% Filtering of faulty data				
	NB	SMO	kNN	C4.5
fanIn	90.44	90.44	90.5	90.44
NAI	90.61	90.6	90.39	90.55
LOC [40]	84.4	82.87	84.56	82.87
NPM[40]	86.4	86.25	86.71	86.17

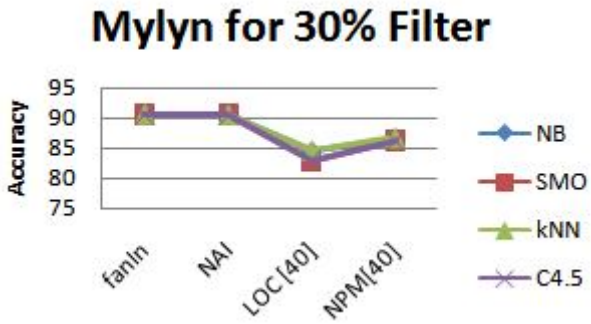

Fig 4: Accuracy graph of Mylyn with 30% filter

In Table 16 and Fig 4, we have shown the accuracy measure for Mylyn with 30% filtering of less complex faulty instances, where we can see that the metrics named fanIn and NAI gave the better results than LOC [40] and NPM [40], for all the four classifiers.

Table 17: Accuracy measure for PDE with No filter

PDE for NO Filter				
	NB	SMO	kNN	C4.5
NPRIM	86.11	86.11	86.04	86.11
LOC [40]	84.7	84.77	86.04	83.9
NPM [40]	84.7	84.77	86.04	83.9

Table 18: Accuracy measure for PDE with 10% filter

PDE for 10% Filtering of faulty data				
	NB	SMO	kNN	C4.5
NPRIM	87.33	87.33	87.26	87.33
LOC [40]	85.07	83.37	84.93	83.37
NPM [40]	86.41	86.19	86.63	86.19

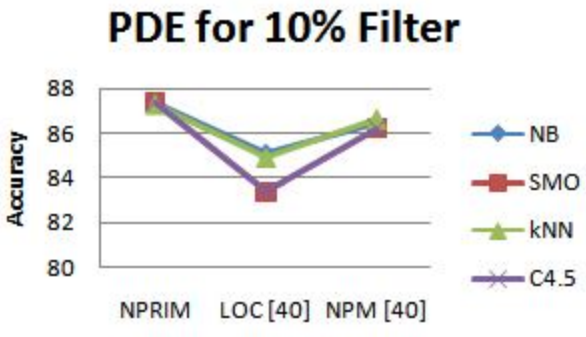

Fig 5: Accuracy graph of PDE with 10% filter

In Table 18 and Fig 5, we have shown the accuracy measure for PDE with 10% filtering of less complex faulty instances, where we can see that the metrics named NPRIM gave the better results than LOC [40] and NPM [40], for all the four classifiers.

Table 19: Accuracy measure for PDE with 20% filter

PDE for 20% Filtering of faulty data				
	NB	SMO	kNN	C4.5
NPRIM	88.66	88.66	88.52	88.66
LOC [40]	84.04	82.2	83.8	82.2
NPM [40]	86.72	86.64	86.97	86.64

Table 20: Accuracy measure for PDE with 30% filter

PDE for 30% Filtering of faulty data				
	NB	SMO	kNN	C4.5
NPRIM	89.96	89.82	89.82	89.82
LOC [40]	82.25	80.82	82.15	80.82
NPM [40]	86.64	86.54	86.73	86.54

In the above result analysis, we have shown the accuracy measures of all the five projects where our performance gave better results than [40]. We have shown the five graphs of for five projects randomly; likewise we can also construct the graphs for all the measures of all the twenty tables shown above. We can analyze from all the tables itself that, the accuracy of our study gave the better results than the work of [40] and can see it graphically by constructing the graphs for the same.

5. CONCLUSION AND FUTURE WORK

We have applied preprocessing technique on the data, attribute selection methodology has been applied to identify the most prominent attributes among all. For this, Wrapper subset evaluation technique has been applied with Naïve Bayes classifier at 10 folds cross validation and 0.05 thresholds as its configuration. Best First search evaluator has been used for searching. After attaining the attributes, we applied 10%, 20% and 30% filtering on the obtained metrics. 10, 20 and 30% of less complex faulty instances are filtered out, which lead to the formation of 3 new datasets for each corresponding metrics. This research uses Naïve Bayes (NB) classifier, Support Vector Machine (SVM) classifier, k- nearest neighbors (kNN) classifier, and C4.5 decision trees classifier for the study. On all these classifiers we have evaluated the accuracy measures against each metric. And we found that the accuracy obtained through our research is much better than the work of [40]. And in future, we wish to expand our study to deal with the class imbalance problem and then perform the same working on the balanced data and analyze their result.

REFERENCES

[1] Gyimothy T, Ferenc R, Siket I, "Empirical validation of object oriented metrics on open source software for fault prediction", IEEE Trans Softw Eng 31(10), pp. 897–910, 2005.

[2] Zhou Y, Leung H, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults". IEEE Trans Softw Eng 32(10), pp. 771–789, 2006.

[3] Shatnawi R, "A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems". IEEE Trans Softw Eng 36(2), pp. 216–225, 2010.

[4] Jureczko M, Madeyski L, "Cross–project defect prediction with respect to code ownership model: an empirical study". e-Inform Softw Eng J 9(1), pp. 21–35, 2015.

[5] Hamill M, Goseva- Popstojanova K, "Exploring the missing link: an empirical study of software fixes". Softw Test Verif Reliab 24(5), pp. 49–71, 2014.

[6] Zhou Y, Xu B, Leung H, Chen L, "An in-depth study of the potentially confounding effect of class size in fault prediction". ACM Trans Softw Eng Methodol 23(1), pp. 1–51, 2014.

[7] Kaur A, Kaur K, Chopra D, "An empirical study of software entropy based bug prediction using machine learning". Int J Syst Assur Eng Manag, pp. 1–18, 2016.

[8] Jindal R, Malhotra R, Jain A, "Prediction of defect severity by mining software project reports". Int J Syst Assur Eng Manag, pp. 1–18, 2016.

[9] Catal C, Alan O, Balkan K, "Class noise detection based on software metrics and ROC curves". Inf Sci 181(21), pp. 4867–4877, 2011.

[10] Seiffert C, Khoshgoftaar TM, Hulse JV, Folleco A, "An empirical study of the classification performance of learners on imbalanced and noisy software quality data". Inf Sci 259, pp. 571–595, 2014.

[11] Hall T, Beecham S, Bowes D, Gray D, Counsell S, "A systematic review of fault prediction performance in

software engineering". IEEE Trans Softw Eng 38(6), pp. 1276–1304, 2011.
[12] Boetticher G "Improving credibility of machine learner models in software engineering". In: Advanced machine learner applications in software engineering, software engineering and knowledge engineering, pp 52–72, 2006.
[13] Schro¨ter A, Zimmermann T, Zeller A, "Predicting component failures at design time". In: Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering. ACM, pp. 18–27, 2006.
[14] Kim S, Zimmermann T, Whitehead E, Zeller A, "Predicting faults from cached history". In: Proceedings of the 29th international conference on software engineering (ICSE), Minneapolis, 20–26 May, 2007.
[15] Jiang Y, Cukic B, Ma Y, "Techniques for evaluating fault prediction models". Empir Softw Eng 13, pp. 561–595, 2008.
[16] Liebchen GA, Shepperd M, "Data sets and data quality in software engineering". Proceedings of the 4th international workshop on predictor models in software engineering (PROMISE '08). ACM, New York, pp. 39–44, 2008.
[17] Gray D, Bowes D, Davey N, Sun Y, Christianson B, "The misuse of the NASA metrics data program data sets for automated software defect prediction". In: Evaluation and assessment in software engineering (EASE), 2011.
[18] Gao K, Khoshgoftaar TM, Seliya N, "Predicting high-risk program modules by selecting the right software measurements". Softw Qual J 20(1), pp. 3–42, 2012.
[19] Al Dallal J, "The impact of accounting for special methods in the measurement of object-oriented class cohesion on refactoring and fault prediction activities". J Syst Softw 85(5), pp. 1042–1057, 2012.
[20] Shepperd M, Song Q, Sun Z, Mair C, "Data quality: some comments on the NASA software defect datasets". IEEE Trans Softw Eng 39(9), pp. 1208–1215, 2013.
[21] Petric´ J, Bowes D, Hall T, Christianson B, Baddoo N "The jinx on the NASA software defect data sets". In: Proceedings of the 20th international conference on evaluation and assessment in software engineering (EASE). Article 13, 5 pages, 2016.
[22] Erni K, Lewerentz C, "Applying design-metrics to objectoriented frameworks". In: Proceedings of the third international software metrics symposium. Pp. 25–26, 1996.
[23] Menzies T, Milton Z, Turhan B, Cukic B, Jiang Y, Bener A, "Defect prediction from static code features: current results, limitations, new approaches". Autom Softw Eng 17, pp. 375–407, 2010.
[24] Available at http://bug.inf.usi.ch. [Accessed: July 15, 2017]
[25] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten I, "The WEKA data mining software, an update". Special Interest Group Knowl Discov Data Min Explor Newsl 11(1), pp. 10–18, 2009.
[26] Menzies T, DiStefano J, Orrego A, Chapman R, "Assessing predictors of software defects". In: Predictive software models workshop, 2004.
[27] Challagulla VU, Bastani FB, Yen I, Paul RA, "Empirical assessment of machine learning based software defect prediction techniques". In: Tenth IEEE international workshop on objectoriented real-time dependable systems, pp. 263–270, 2005.
[28] John GH, Langley P, "Estimating continuous distributions in Bayesian classifiers". In: Besnard P, Hanks S (eds) Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, pp. 338–345, 1995.
[29] Aha D, Kibler D, "Instance-based learning algorithms". Mach Learn 6(1), pp. 37–66, 1991.
[30] Wang H, Khoshgoftaar TM, Seliya N, "How many software metrics should be selected for defect prediction?" In: Murray RC, McCarthy PM (Eds) FLAIRS Conference. AAAI Press, Palo Alto, 2011.
[31] Gao K, Khoshgoftaar K, Wang H, Seliya N, " Choosing software metrics for defect prediction": an investigation on feature selection techniques. Softw Pract Exp 41(5), pp. 579–606, 2011.
[32] Quinlan JR, "C4.5: Programs for machine learning". Morgan Kaufmann Publishers, San Mateo, 1993.
[33] Mertik M, Lenic M, Stiglic G, Kokol P "Estimating software quality with advanced data mining techniques". In: International conference on software engineering advances. p 19, 2006.
[34] Riquelme JC, Ruiz R, Rodrı´guez D, Moreno J, "Finding defective modules from highly unbalanced datasets". Actas del 8_taller sobre el apoyo a la decisio´n en ingenieri´a del software 2(1), pp. 67–74, 2008.
[35] http://home.iitk.ac.in/~arunothi/pclub/ml/3.pdf. [Accessed: September 21, 2017]
[36] http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html. [Accessed: September 21, 2017]
[37] http://www.technology.heartland.edu/courses/Computer%20Science/Programming/Java%20Courses/JTest%20User%27s%20Guide/metnprim.htm. [Accessed: September 21, 2017]
[38] http://support.objecteering.com/objecteering6.1/help/us/metrics/metrics_in_detail/number_of_attributes.htm. [Accessed: September 21, 2017]
[39] https://www.researchgate.net/post/How_do_I_measure_FAN-OUT_FAN-IN. [Accessed: September 21, 2017]
[40] Raed Shatnawi, "Identifying and eliminating less complex instances from software fault data". In: Int J Syst Assur Eng Manag, Springer, 2016.

AUTHOR

Prabujeet Kaur is a student of M.Tech, in the Department of Computer Science and Engg, Kamla Nehru Institute of Technology (KNIT), Sultanpur, UP, India. She has received a B.Tech degree (2012) from Sri Ram Murti Smarak Women's College of Engg and Tech, Bareilly, UP, India. She has a work experience of a Guest Lecturer for 2 and a half year from KNIT, Sultanpur, UP, India. She has

published 2 papers till now. Her research interests are Software Engineering, and Cryptography & Network Security.

Dharmendra Lal Gupta is currently working as an Associate Professor in the Department of Computer Science & Engineering at Kamla Nehru Institute of Technology (KNIT), Sultanpur (U.P.) India. He received B.Tech. (1999) from KNIT, Sultanpur, (U.P.), in Computer Science & Engineering, M.Tech. Hon's (2003) in Digital Electronics and Systems from KNIT, Sultanpur, (U.P.), India. He has been a member of IEEE Computer Society. He has published about 26 papers in International/National Journals/workshops/conferences and seminars. His research interests are Software Quality Engineering, Software Engineering, and Cryptography & Network Security.