EXPERIMENTAL ANALYSIS OF OPEN SOURCE PROJECTS FOR COST ESTIMATION USING DATA PRE-PROCESSING AND LEARNING TECHNIQUES

[1]Prabujeet Kaur, [2]Dharmendra Lal Gupta
[1]M.Tech Scholar, [2]Associate Professor
[1]Department of Computer Science and Engineering, Kamla Nehru Institute of Technology, Sultanpur, Uttar Pradesh, India
[2]Department of Computer Science and Engineering, Kamla Nehru Institute of Technology, Sultanpur, Uttar Pradesh, India

Abstract: To achieve software quality for large systems is very difficult. Developers and testers put a lot of their effort to evaluate the software quality which turns out to be very time consuming process. The software quality can be accessed through fault prediction or cost estimation and many more. Various studies have been carried out for various kinds of prediction processes. In each of which, machine learning techniques are used for prediction purposes. In this research, feature selection and data preprocessing techniques has been carried out. In this Wrapper subset evaluation method has been chosen for attribute selection. After attribute selection process, 10, 20 and 30 % of less complex faulty instances were filtered out form each selected attribute. Later, the resultant datasets were processed against four classifiers: Naïve Bayes, Support Vector Machine, k nearest neighbors and C4.5 Decision trees. Cost estimation against each attribute was calculated. Lastly, the calculated result was compared against the cost estimation of filtering out of less complex instances for LOC and NPM metrics. And through comparison, the research show that the Classifiers based on Wrapper subset evaluation method gave better results than filtering out of less complex instances for LOC and NPM metrics.

Index Terms: **Software fault; cost estimation; data preprocessing; feature selection; complexity**

I. INTRODUCTION

Software metrics help the researchers or the users to identify the quality of the software. The software costs increases drastically when the software is of good quality. Software project managers must assess the cost or effort needed for creating the software at a beginning time of its life-cycle [42]. The capacity to precisely evaluate the development cost assumes an imperative part in the success of software systems. Software engineers utilize these metrics to investigate whether the quality can be enhanced. Mostly all software systems are bigger in size where some classes are large and some classes are small to smaller in size and complexity. And small classes may likewise require less time for exploring their quality. Therefore, software engineers need to distribute their assets or resources effectively just to those parts of the software's which require more efforts.

Various analysis tools and metrics data are available easily. Data mining is a strategy which is utilized to group the modules or instances into defective or not defective through metric value [1, 2, 3, 4, 5, 6, 7 and 8]. Software quality assessed by utilizing different data mining tools may not generally give great outcome if the quality of the data is low like noise [9] and class imbalance [10]. Hence, at times preprocessing is required before predicting the nature or the quality of software.

In this research, most conspicuous metrics are chosen through data pre-processing. Furthermore, for this, Wrapper subset technique has been utilized which came about into various different metric subsets for different projects. Further, faulty instances which were less complex were filtered out and removed from each dataset. Three filters were proposed for removal viz. 10, 20 and 30% of less complex faulty data. The general structure of the paper is: Section 2 talks about the related work for cost estimation in software. Section 3 talks about the research methodology utilized for this research. Section 4 speaks
about the outcomes of this research on four classification algorithms and also the correlation with the performance of [40] has also been discussed. Finally, Section 6 prompts conclusion and future extent of the research.

II. REALTED WORK

Fault prediction and Cost Estimation models confront numerous troubles like data quality and in addition class imbalance issue [11]. Therefore, many researchers have presented diverse techniques for data preprocessing which can enhance the prediction procedure.

Boetticher G [12] applied data pre-processing by the removal of replicated instances from NASA datasets.

Schro¨ter A, Zimmermann T and Zeller A [13] took 52 diverse ECLIPSE modules, led data pre-processing and chose the dataset from the defective parts as it were.

Kim S, Zimmermann T, Whitehead E and Zeller A [14], the authors presumed that lone 10% of modules represent over 73% of defects in seven open-source projects.

Jiang Y, Cukic B and Ma Y [15] have tried the effect of two procedures, log and discretization change on ten classifier algorithms. In any case, the authors couldn't locate any dominant method.

In another investigation, Gyimothy T, Ferenc R and Siket I [1] has recognized a relationship between the most basic parts of the code and cost of testing these parts utilizing various models.

Liebchen GA and Shepperd M [16] have revealed that exclusively 23 out of 100 fault prediction studies thinks about the quality of data while numerous models were built without data cleaning (Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P and Witten I [25]).

Gray D, Bowes D, Davey N, Sun Y and Christianson B [17] have led data pre-processing on NASA datasets and have removed 6 to 90% of the original data utilizing different cleaning procedures.

Catal C, Alan O and Balkan K [9] have utilized thresholds system to distinguish two kinds of instances as noisy. A non-defective instance is noisy if metric values are greater than their relating thresholds. A defective instance is noisy if metric values are not as much as their relating thresholds.

Gao K, Khoshgoftaar TM and Seliya N [18] conducted an empirical analysis by utilizing various sampling methodologies thereafter utilized feature selection methodology so as to enhance the effectiveness of the prediction processes.

Al Dallal J [19] has examined the impact of extraordinary techniques like constructors, destructors, and access methods for estimating the cohesion of classes. The outcomes indicated critical contrasts in cohesion measurements yet there were no huge impacts on fault prediction process.

Shepperd M, Song Q, Sun Z and Mair C [20] have completed 18 referential integrity checks for data validity and discovered tremendous measure of blunders in data. Data were ordered into problematic data and non problematic yet that does not help in fault prediction processes. Problematic data have impossible values and non problematic data have repeated attributes.

Petric´ J, Bowes D, Hall T, Christianson B and Baddoo N [21] presented another checks for data integrity so as to clean NASA datasets. They included two integrity checks along with the work of [20]; however, the authors have not done any fault prediction although.

Erni K and Lewerentz C [22] proposed the usage of mean and standard deviation in order to find out two possible threshold values, the minimum threshold viz. Tmin and the maximum threshold viz. Tmax. These threshold values are calculated as follows, Tmin = l – s and Tmax = l ? s, being l the average of a metric and s the standard deviation

Jianglin Huang, Yan-Fu Li and Min Xie [41] contemplated the literature survey of data pre-processing procedures initially. Thusly, an experimental analysis led to break down the effectiveness of 4 data pre-processing strategies. ANOVA test is led to evaluate the hugeness of each pre-processing procedure and the interactions amongst them and machine learning techniques.

In this research, filtering was carried out to remove the less complex faulty instances out of the original data and have proved to be cost-effective as well.

III. RESEARCH METHODOLOGY

Data quality is very important to enhance the prediction processes. An empirical investigation has been applied so as to watch the effect of data pre-processing on the performance of fault prediction and cost estimation models.

3.1 Data Sources

This research involves different open source projects. They are available at [24] publically:

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%

3.2 Feature Selection

Before applying any data preprocessing technique, faeture selection has been done to identify the metrics which are more prominent amongst all. For feature selection, Wrapper subset technique has been used with configuration as Naïve Bayes classifier at 10 folds and 0.05 thresholds. And for searching, Best First search technique has been used. The resultant metrics were:

3.2.1 ***CBO:*** It is an aggregation of classes that are coupled to a solitary class [36].

3.2.2 ***NOA:*** It is an aggregation of attributes in a class; and at package level it is an aggregate number of attributes per class [38].

3.2.3 ***NMI:*** **I**t is an aggregation of methods that are acquired by the child class from the parent class [37].

3.2.4 ***DIT:*** It is the greatest length from the hub or root of a tree to the node of a tree and it can be estimated as the aggregation of ancestral classes [36].

3.2.5 ***NOC:*** It is an aggregation of quick sub-classes of a class [36].

3.2.6 ***NAI:*** It is the aggregation of attributes that are acquired by the child class from the parent class [37].

3.2.7 ***NPRIM:*** It is an aggregation of methods that are declared inside a class [37].

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

3.2.9 ***FAN-IN:*** It is an aggregation of methods that call some other method [39].

3.2.10 ***FAN-OUT:*** It is an aggregation of methods that are called by another method [39].

3.3 Data Filtering

After getting all the features or metrics, we filtered out 10%, 20% and 30% of less complex faulty instances from each metric, which resulted into 3 new datasets for each metric correspondingly.

3.4 Classification techniques and performance evaluation

The research includes four specific classifier algorithms viz. Naïve Bayes (NB), Support Vector Machine (SVM), k- nearest neighbors (kNN), and C4.5 decision trees. Weka tool has been used for training and testing these classifiers [25].

Naive Bayes (NB) classification algorithm is widely used for prediction processes [26, 27]. It makes use of Bayesian network that follows two assumptions. Firstly, all the metrics are independent completely where as the classes may be defective or non-defective and secondly, hidden attributes can not affect the prediction method [28].

The SVM classification algorithm is a binary algorithm that keeps the margin at its maximum limit. The separator also called hyper-plane. It is parallel and midway between the margin planes. Each margin plane goes through point(s) that have a place with a specific class and is nearest to the margin plane of alternate class. The separation between these margin planes is known as margin. One thing to make a note here is that numerous sets of margin planes can be possible with various margins. In any case, SVM finds the margin which is at its most extreme point of confinement from both the sides of the hyper-plane. The points from each class that go through the margin planes and are named as support vectors [35].

The k nearest neighbors (kNN) classification algorithm measures the separation or similarity between the modules utilizing metric values and allocate modules to be either defective or non defective as indicated by the dominancy of the nearest group of nodes [29]. The K value is generally set to be an odd and this research, uses k = 5. The k nearest neighbor's classifier algorithm has been utilized in various previous researches for prediction purposes [2, 30 and 31].

C4.5 decision tree classification algorithm uses information based approach viz. information gain to build the tree [32]. The tree develops by choosing the metric value with the highest information. C4.5 decision tree classifier algorithm has been utilized in various researches for prediction processes [33 and 34]. All the classification algorithms makes use of 10 fold cross-validations [40].

3.5 Performance measures chosen for this research is:

3.5.1 *Cost Estimation*: False_Positives+True_Negatives

It can also be calculated from the confusion matrix.

IV. RESULT ANALYSIS

This analysis shows the measures for cost estimation in all the five projects at No filter, 10% filter, 20% filter and 30% filter, which removes less complex faulty instances from the original data. The result and performance measure are discussed below.

Table 1: Cost Estimation measure of Eclipse JDT with No Filter

ECLIPSE JDT FOR NO FILTER				
	NB	***SMO***	***kNN***	***C4.5***
CBO	164	171	206	198
NOA	164	171	206	198
NMI	164	171	206	198
LOC [40]	211	213	211	216
NPM [40]	211	213	211	216

Table 2: Cost Estimation measure of Eclipse JDT with 10% filter

Eclipse JDT for 10% Filtering of faulty data				
	NB	SMO	kNN	C4.5
CBO	159	160	189	163
NOA	182	182	186	182
NMI	178	178	186	178
LOC [40]	205	207	193	207
NPM [40]	182	182	194	182

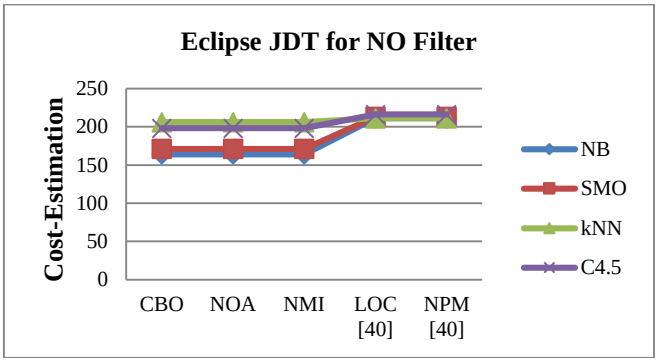

Figure 1: Cost Estimation graph of Eclipse JDT with No Filter

In Table 1 and Figure 1 above, it shows that the Cost Estimation for Eclipse JDT with No filtering where CBO, NOA and NMI gave much better result than LOC [40] and NPM [40] for all the four classification algorithms. Similarly the graphs for other Eclipse JDT filters can also be constructed through the values in the tables below.

Table 3: Cost Estimation measure of Eclipse JDT with 20% filter

ECLIPSE JDT FOR 20% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
CBO	150	145	172	150
NOA	159	159	165	159
NMI	161	155	162	155
LOC [40]	198	200	187	200
NPM [40]	167	170	181	170

Table 4: Cost Estimation measure of Eclipse JDT with 30% filter

ECLIPSE JDT FOR 30% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
CBO	129	125	154	133
NOA	138	141	147	141
NMI	142	140	147	140
LOC [40]	197	198	185	198
NPM [40]	150	150	162	150

Table 5: Cost Estimation measure of Equinox with No filter

EQUINOX FOR NO FILTER				
	NB	***SMO***	***KNN***	***C4.5***
DIT	95	108	118	119
NOC	95	108	118	119
NAI	95	108	118	119
NPRIM	95	108	118	119
NPM	95	108	118	119
LOC [40]	96	107	124	123
NPM [40]	96	107	124	123

Table 6: Cost Estimation measure of Equinox with 10% filter

EQUINOX FOR 10% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
DIT	108	108	108	108
NOC	108	110	107	108
NAI	114	109	121	114
NPRIM	113	100	118	115
NPM	114	104	117	113

LOC [40]	120	91	126	120
NPM [40]	113	102	117	113

Table 7: Cost Estimation measure of Equinox with 20% filter

EQUINOX FOR 20% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
DIT	90	90	90	90
NOC	94	97	94	94
NAI	98	93	107	98
NPRIM	104	91	111	105
NPM	97	83	98	95
LOC [40]	118	88	125	118
NPM [40]	113	96	116	113

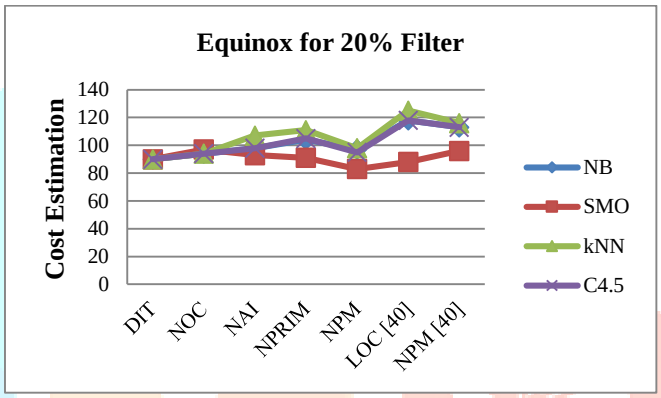

Figure 2: Cost Estimation graph of Equinox with 20% filter

In Table 7 and Figure 2 above, it shows that the Cost Estimation for Equinox with 20% filtering out of less complex faulty instances where DIT, NOC, NAI, NPRIM and NPM gave much better result than LOC [40] and NPM [40] for all the four classification algorithms. Similarly the graphs for other Equinox filters can also be constructed through the values in the tables below.

Table 8: Cost Estimation measure of Equinox with 30% filter

EQUINOX FOR 30% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
DIT	77	77	77	77
NOC	81	83	81	81
NAI	85	80	94	85
NPRIM	90	76	96	91
NPM	89	65	86	83
LOC [40]	82	82	104	86
NPM [40]	104	104	105	111

Table 9: Cost Estimation measure of Lucene with No filter

LUCENE FOR NO FILTER				
	NB	***SMO***	***kNN***	***C4.5***
fanOut	62	62	64	62
LOC [40]	71	72	64	74
NPM [40]	71	72	64	74

Table 10: Cost Estimation measure of Lucene with 10% filter

LUCENE FOR 10% FILTERING OF FAULTY DATA

	NB	SMO	kNN	C4.5
fanOut	54	54	58	54
LOC [40]	61	68	61	68
NPM [40]	62	62	60	62

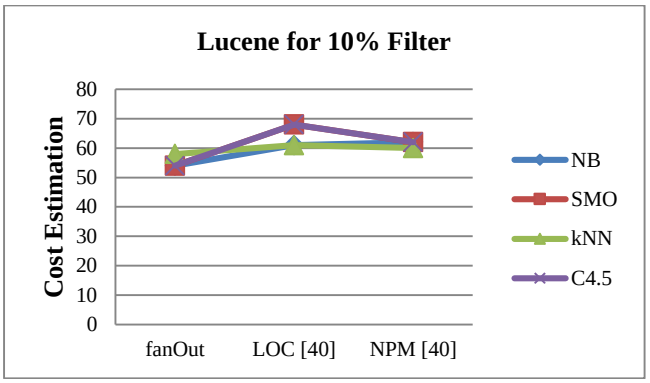

Figure 3: Cost Estimation graph of Lucene with 10% filter

In Table 10 and Figure 3 above, it shows that the Cost Estimation for Lucene with 10% filtering out of less complex faulty instances where fanOut gave much better result than LOC [40] and NPM [40] for all the four classification algorithms. Similarly the graphs for other Lucene filters can also be constructed through the values in the tables below.

Table 11: Cost Estimation measure of Lucene with 20% filter

LUCENE FOR 20% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
fanOut	47	47	51	47
LOC [40]	58	70	58	70
NPM [40]	59	59	56	59

Table 12: Cost Estimation measure of Lucene with 30% filter

LUCENE FOR 30% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
fanOut	43	45	45	45
LOC [40]	56	65	56	65
NPM [40]	54	54	51	54

Table 13: Cost Estimation measure of Mylyn with No filter

MYLYN FOR NO FILTER				
	NB	***SMO***	***kNN***	***C4.5***
fanIn	240	241	240	242
NAI	240	241	240	242
LOC [40]	276	272	248	275
NPM [40]	276	272	248	275

Table 14: Cost Estimation measure of Mylyn with 10% filter

MYLYN FOR 10% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
fanIn	221	221	216	221
NAI	217	217	221	217
LOC [40]	239	249	232	249
NPM [40]	228	228	228	228

Table 15: Cost Estimation measure of Mylyn with 20% filter

MYLYN FOR 20% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
fanIn	194	194	191	194

NAI	194	194	196	194
LOC [40]	216	239	212	239
NPM [40]	204	204	200	204

Table 16: Cost Estimation measure of Mylyn with 30% filter

MYLYN FOR 30% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
fanIn	172	172	168	172
NAI	168	168	172	168
LOC [40]	201	223	204	223
NPM [40]	177	179	275	179

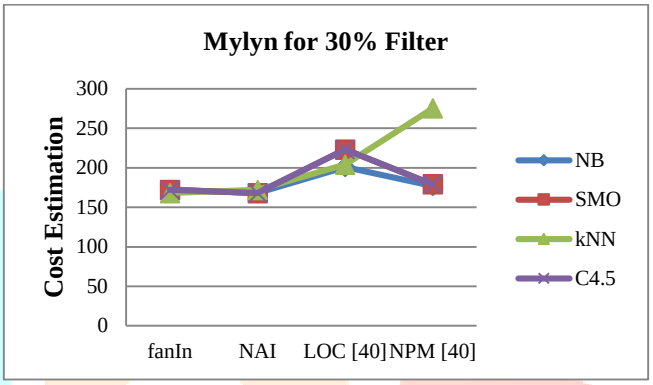

Figure 4: Cost Estimation graph of Mylyn with 30% filter

In Table 16 and Figure 4 above, it shows that the Cost Estimation for Mylyn with 30% filtering out of less complex faulty instances where fanIn and NAI gave much better result than LOC [40] and NPM [40] for all the four classification algorithms. Similarly the graphs for other Mylyn filters can also be constructed through the values in the tables below.

Table 17: Cost Estimation measure of PDE with No filter

PDE FOR NO FILTER				
	NB	***SMO***	***KNN***	***C4.5***
NPRIM	209	209	209	209
LOC [40]	231	228	212	240
NPM [40]	231	228	212	240

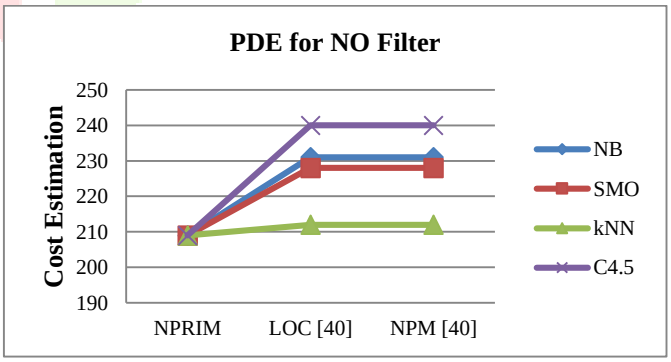

Figure 5: Cost Estimation graph of PDE with No filter

In Table 17 and Figure 5 above, it shows that the Cost Estimation for PDE with No filtering where NPRIM gave much better result than LOC [40] and NPM [40] for all the four classification algorithms. Similarly the graphs for other PDE filters can also be constructed through the values in the tables below.

Table 18: Cost Estimation measure of PDE with 10% filter

PDE FOR 10% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
NPRIM	187	187	188	187
LOC [40]	209	218	201	218

NPM [40]	186	186	182	186

Table 19: Cost Estimation measure of PDE with 20% filter

PDE FOR 20% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
NPRIM	166	166	167	166
LOC [40]	196	211	193	211
NPM [40]	162	162	158	162

Table 20: Cost Estimation measure of PDE with 30% filter

PDE FOR 30% FILTERING OF FAULTY DATA				
	NB	***SMO***	***kNN***	***C4.5***
NPRIM	147	150	149	150
LOC [40]	188	207	186	207
NPM [40]	144	144	141	144

In the above result analysis, we have shown the Cost Estimation measures of all the five projects. We have shown the measure of Cost Estimation for each project in a tabular form and also have constructed a graph of five tables randomly (one for each kind of project). Likewise the graphs for all the measures of all the 16 remaining tables can be constructed through the values in the tables above. One can analyze from all the tables itself that the Cost Estimation measure of our study gave the better results in most of the cases than the work of [40] and can see it graphically by constructing the graphs for the same.

V. CONCLUSION AND FUTURE SCOPE

Feature selection technique has been applied before data pre-processing. Feature selection has been applied to identify the features or metrics which are more prominent amongst all. For this, Wrapper subset technique has been applied with its configuration as Naïve Bayes classification algorithm at 10 folds and 0.05 thresholds. For searching purposes, Best First search method has been utilized. Data pre-processing has been done on the resultant metrics of the feature selection method. In this, filtering out of 10%, 20% and 30% of less complex faulty instances on each metric, which resulted into the creation of 3 new datasets for each metric. Further, this research uses four classification algorithm viz. Naïve Bayes (NB), Support Vector Machine (SVM), k- nearest neighbors (kNN), and C4.5 decision trees for carrying out the prediction of cost. On all these classifiers we have evaluated the Cost Estimation measures against each metric. And it has been found that the Cost estimation obtained through our research is much better than the work of [40].

And in future, we wish to expand our study to deal with class imbalance issue and then perform the same working on the balanced data and analyze that result.

VI. REFERENCES

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

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

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

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

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

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

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

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

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

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

[11] Hall T, Beecham S, Bowes D, Gray D, Counsell S, 2011. A systematic review of fault prediction performance in software engineering. IEEE Trans Softw Eng. Volume 38(6), pp. 1276–1304.

[12] Boetticher G, 2006. 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.

[13] Schro¨ter A, Zimmermann T, Zeller A, 2006. Predicting component failures at design time. In: Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering. ACM, pp. 18–27.

[14] Kim S, Zimmermann T, Whitehead E, Zeller A, 2007. Predicting faults from cached history. In: Proceedings of the 29th international conference on software engineering (ICSE), Minneapolis, 20–26 May.
[15] Jiang Y, Cukic B, Ma Y, 2008. Techniques for evaluating fault prediction models. Empir Softw Eng. Volume 13, pp. 561–595.
[16] Liebchen GA, Shepperd M, 2008. 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.
[17] Gray D, Bowes D, Davey N, Sun Y, Christianson B, 2011. The misuse of the NASA metrics data program datasets for automated software defect prediction. In: Evaluation and assessment in software engineering (EASE).
[18] Gao K, Khoshgoftaar TM, Seliya N, 2012. Predicting highrisk program modules by selecting the right software measurements. Softw Qual J. Volume 20(1), pp. 3–42.
[19] Al Dallal J, 2012. The impact of accounting for special methods in the measurement of object-oriented class cohesion on refactoring and fault prediction activities. J Syst Softw. Volume 85(5), pp. 1042–1057.
[20] Shepperd M, Song Q, Sun Z, Mair C, 2013. Data quality: some comments on the NASA software defect datasets. IEEE Trans Softw Eng. Volume 39(9), pp. 1208–1215.
[21] Petric´ J, Bowes D, Hall T, Christianson B, Baddoo N, 2016. 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.
[22] Erni K, Lewerentz C, 1996. Applying design-metrics to object-oriented frameworks. In: Proceedings of the third international software metrics symposium. Pp. 25–26.
[23] Menzies T, Milton Z, Turhan B, Cukic B, Jiang Y, Bener A, 2010. Defect prediction from static code features: current results, limitations, new approaches. Autom Softw Eng. Volume 17, pp. 375–407.
[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, 2009. The WEKA data mining software, an update. Special Interest Group Knowl Discov Data Min Explor Newsl. Volume11(1), pp. 10–18.
[26] Menzies T, DiStefano J, Orrego A, Chapman R, 2004. Assessing predictors of software defects. In: Predictive software models workshop.
[27] Challagulla VU, Bastani FB, Yen I, Paul RA, 2005. Empirical assessment of machine learning based software defect prediction techniques. In: Tenth IEEE international workshop on objectoriented real-time dependable systems, pp. 263–270.
[28] John GH, Langley P, 1995. 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.
[29] Aha D, Kibler D, 1991. Instance-based learning algorithms. Mach Learn. Volume 6(1), pp. 37–66.
[30] Wang H, Khoshgoftaar TM, Seliya N, 2011. How many software metrics should be selected for defect prediction? In: Murray RC, McCarthy PM (Eds) FLAIRS Conference. AAAI Press, Palo Alto.
[31] Gao K, Khoshgoftaar K, Wang H, Seliya N, 2011. Choosing software metrics for defect prediction: an investigation on feature selection techniques. Softw Pract Exp. Volume 41(5), pp. 579–606.
[32] Quinlan JR, 1993. C4.5: Programs for machine learning. Morgan Kaufmann Publishers, San Mateo.
[33] Mertik M, Lenic M, Stiglic G, Kokol P, 2006. Estimating software quality with advanced data mining techniques. In: International conference on software engineering advances. p 19.
[34] Riquelme JC, Ruiz R, Rodrı´guez D, Moreno J, 2008. Finding defective modules from highly unbalanced datasets. Actas del 8_taller sobre el apoyo a la decisio´n en ingenierı´a del software. Volume 2(1), pp. 67–74.
[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%20ser%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, 2016. Identifying and eliminating less complex instances from software fault data. In: Int J Syst Assur Eng Manag, Springer.
[41] Jianglin Huang, Yan-Fu Li, Min Xie, 2015. An empirical analysis of data preprocessing for machine learning-based software cost estimation. Information and Software Technology. Volume (67), pp. 108-127.
[42] J. Wen, S. Li, Z. Lin, Y. Hu, C. Huang, 2012. Systematic literature review of machine learning based software development effort estimation models. Inform. Softw. Technol. Volume (54) pp. 41–59.