Analyzing Fault Prediction Usefulness from Cost Perspective using Source Code Metrics

Lov Kumar
NIT Rourkela, India
lovkumar505@gmail.com

Ashish Sureka
ABB Corporate Research, India
ashish.sureka@in.abb.com

***Abstract*—Software fault prediction techniques are useful for the purpose of optimizing test resource allocation. Software fault prediction based on source code metrics and machine learning models consists of using static program features as input predictors to estimate the fault proneness of a class or module. We conduct a comparison of five machine learning algorithms on their fault prediction performance based on experiments on 56 open source projects. Several researchers have argued on the application of software engineering economics and testing cost for the purpose of evaluating a software quality assurance activity. We evaluate the performance and usefulness of fault prediction models within the context of a cost evaluation framework and present the results of our experiments. We propose a novel approach using decision trees to predict the usefulness of fault prediction based on distributional characteristics of source code metrics by fusing information from the output of the fault prediction usefulness using cost evaluation framework and distributional source code metrics.**

Keywords*—*Software Engineering, Source Code Metrics, Fault Prediction, Machine Learning, Software Engineering Economics

I. Introduction

Software fault prediction or fault-proneness prediction consists of using various types of indicators as features in a statistical model to estimate the fault-proneness of a module or a class [1][2][3]. Fault prediction models are useful for estimating the quality of a software and optimizing test resource allocation. The predictors and indicators for fault proneness can be derived from various software repositories such as source code, version control system and issue tracking system. Several studies are conducted on the application of source code metrics and machine learning algorithms for fault-proneness prediction [1][2][3] as well as a change-proneness prediction [4][5] (both of which are motivated by the need to optimize testing efforts). Several researchers present the application of cost and economics for the purpose of evaluating a fault prediction model in addition to the standard metrics such as accuracy and f-measure [3][6][7][8]. The cost of defect detection, bug fixing, testing tool set-up and configuration cost as well testing efficiency can vary across software development life-cycle phase and characteristics of the project. Hence incorporating cost and economic factors in fault prediction model evaluation is a more robust and practical approach.

For example, Jiang et al. emphasize the importance of cost while evaluating fault prediction models [6][7]. They mention that the cost of false positive (predicting that non-faulty module has a fault) and false negative (predicting a faulty module is non-faulty) of a fault prediction model cannot be regarded as the same [6][7]. The cost of these two different types of misclassification may vary for different projects and hence should be incorporated in a fault prediction evaluation framework. Similarly, Wagner et al. discuss return on investment for a fault-detection method [8]. They specify defect removal cost of a testing technique in-terms of the staff-hours per fault across different types of testing such as system, integration and unit testing [8].

There has been a lot of work done in the area of fault-proneness prediction using source code metrics and machine learning models. However, evaluating the usefulness of fault-proneness prediction using cost factors is an area which is relatively unexplored consisting of a small number of studies. The study presented in this paper makes several novel contributions in context to the body of work on fault proneness-prediction. We conduct experiments on 56 open-source software projects using 5 machine learning classifiers for building fault prediction models. We apply a cost evaluation framework based on the concepts and empirical data from previous studies and present a method to determine when a fault prediction model is economically useful in terms of a cost benefit analysis. We propose a novel approach to predict fault proneness using the distributional characteristics of source code by building a decision tree classifier from data obtained from the output of the fault prediction usefulness and statistical characteristics of source code metrics.

II. Research Framework and Solution Approach

Figure 1 and 2 illustrates the research framework and the proposed solution approach. The dataset downloaded by us from the PROMISE repository consists of the 20 Chidamber and Kemerer source code metrics values for all the classes of the software system. The 20 source code metrics are AMC, IC, DIT, RFC, CBO, LCOM, NOC, Ca, Ce, LCOM3, AVG-CC, NPM, DAM, MOA, LOC, CAM, MFA, CBM, MAX-CC, WMC [9]. Chidamber and Kemerer Java Metrics consists of metrics such as Lack of cohesion in methods (LCOM), Coupling between object classes (CBO), Depth of Inheritance Tree (DIT), Weighted methods per class (WMC) and Afferent couplings (Ca) which have been proven to be correlated with fault-proneness [10].

As shown in Figure 1, we apply 5 different machine learning techniques to build fault-prediction models. Malhotra et al. analyze several papers published in the year 1991 - 2013 and present a literature survey on machine learning techniques for software fault prediction [11]. Challagulla et al. conduct an evaluation of several machine learning based defect predictors

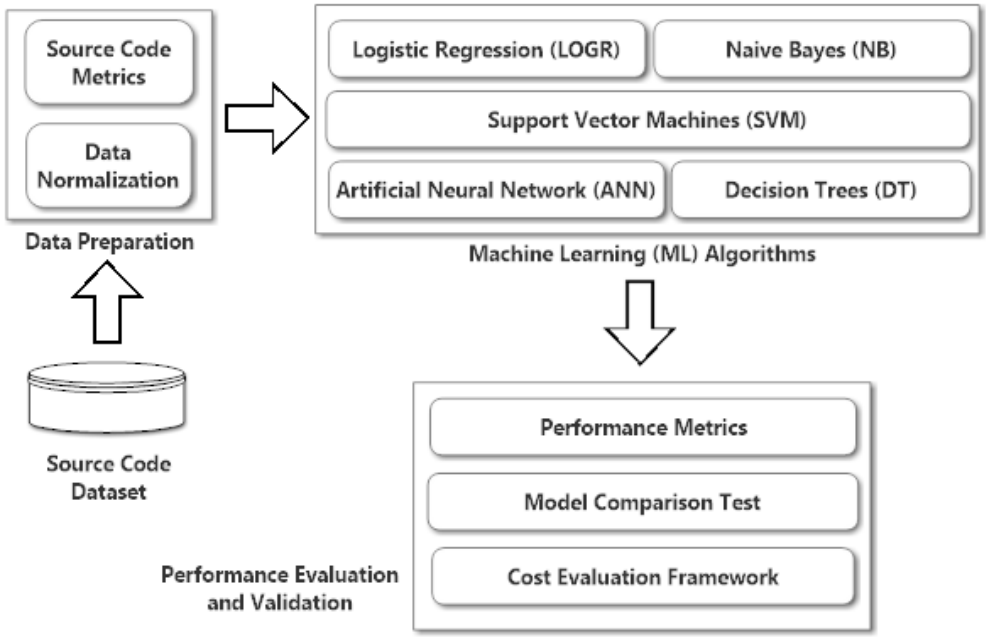

Fig. 1. Architecture Diagram Illustrating Fault Prediction Model Building Using Machine Learning Algorithms and Evaluation Using Statistical Tests

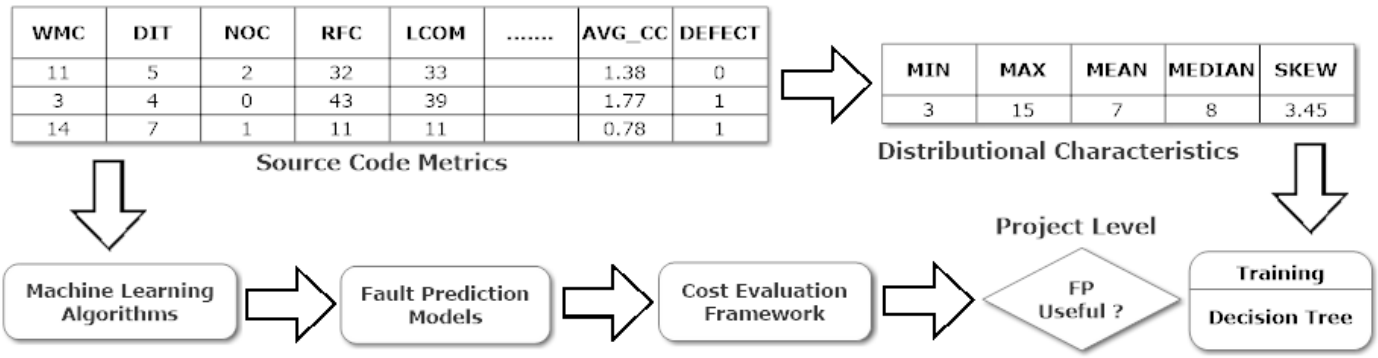

Fig. 2. Architecture Diagram and Multi-Step Process Defining the Sequence of Operations to Build a Fault Prediction Model Using Distributional Characteristics of Source Code

on four real-time software applications [12]. Gondra et al. conduct experiments on NASA data and evaluate algorithms such as ANN and SVM. The 56 project dataset used in our experiments is different than previous work and our work is the first study on the cost evaluation framework in the context of fault prediction for the 56 project dataset.

We evaluate the performance of fault prediction model using metrics such as accuracy and f-measure. We conduct statistical tests such as Wilcoxon signed rank test using Bonferroni correction to test the null hypothesis and determine of the accuracy of the fault prediction models is a function of the classifier characteristics. We use the cost evaluation framework defined by Kumar et al. and Wagner et al. [3][8]. According to the cost evaluation framework by Kumar et al. NE_{cost} represents the normalized fault removal cost and is equal to the ratio of the estimated cost using fault prediction model and without fault prediction model. If the value of NE_{cost} is greater than 1 then the fault removal cost is useful. δ_u, δ_i, and δ_s represents the testing efficiency for unit, integration and system testing respectively. These symbols (NE_{cost}, δ_u, δ_i, and δ_s) are used in the experimental analysis and results section of this paper. Refer to the work by Kumar et al. and Wagner et al. for the complete derivation and reference or benchmark values for these variables [3][8]. Figure 2 illustrates our novel approach for building a fault prediction usefulness model incorporating cost factors from distributional characteristics of source code metrics. As show in Figure 2, our approach is a multi-step process consist of creating an annotated data of fault prediction usefulness for 56 projects and then using it to build a decision tree classifier using distribution characteristics of source code metrics as attributes of the tree.

III. Experimental Dataset

We conduct experiments on dataset downloaded from the PROMISE[1] repository [13]. The PROMISE repository consists of software engineering research dataset which can be freely downloaded. Hence our dataset is publicly available and our

[1]url http://openscience.us/repo/defect/

TABLE I. EXPERIMENTAL DATASET DOWNLOADED FROM PROMISE REPOSITORY. PID : PROJECT ID, # CLS : NUMBER OF CLASSES, # FLC : NUMBER OF FAULTY CLASSES, % FAULTY : PERCENTAGE OF FAULTY CLASSES

PID	Project	# CLS	# FLC	% Faulty	PID	Project	# CLS	# FLC	% Faulty
P1	ant-1.3	125	20	16	P29	nieruchomosci	27	10	37.04
P2	ant-1.4	178	40	22.47	P30	pdftranslator	33	15	45.45
P3	ant-1.5	293	32	10.92	P31	prop-1	18471	2738	14.82
P4	ant-1.6	351	92	26.21	P32	prop-2	23014	2431	10.56
P5	ant-1.7	745	166	22.28	P33	prop-3	10274	1180	11.49
P6	arc	234	27	11.54	P34	prop-4	8718	840	9.64
P7	berek	43	16	37.21	P35	prop-5	8516	1299	15.25
P8	camel-1.0	339	13	3.83	P36	prop-6	660	66	10
P9	camel-1.2	608	216	35.53	P37	redaktor	176	27	15.34
P10	camel-1.4	872	145	16.63	P38	serapion	45	9	20
P11	camel-1.6	965	188	19.48	P39	skarbonka	45	9	20
P12	e-learning	64	5	7.81	P40	sklebagd	20	12	60
P13	intercafe	27	4	14.81	P41	synapse-1.0	157	16	10.19
P14	ivy-1.1	111	63	56.76	P42	synapse-1.1	222	60	27.03
P15	ivy-1.4	241	16	6.64	P43	synapse-1.2	256	86	33.59
P16	ivy-2.0	352	40	11.36	P44	systemdata	65	9	13.85
P17	jedit-3.2	272	90	33.09	P45	szybkafucha	25	14	56
P18	jedit-4.0	306	75	24.51	P46	termoproject	42	13	30.95
P19	jedit-4.1	312	79	25.32	P47	tomcat	858	77	8.97
P20	jedit-4.2	367	48	13.08	P48	velocity-1.5	214	142	66.36
P21	jedit-4.3	492	11	2.24	P49	velocity-1.6	229	78	34.06
D22	kalkulator	27	6	22.22	D50	workflow	39	20	51.28
P23	log4j-1.0	135	34	25.19	P51	wspomaganiepi	18	12	66.67
P24	log4j-1.1	109	37	33.94	P52	xerces-1.2	440	71	16.14
P25	log4j-1.2	205	189	92.2	P53	xerces-1.3	453	69	15.23
P26	lucene-2.0	195	91	46.67	P54	xerces-1.4	588	437	74.32
P27	lucene-2.2	247	144	58.3	P55	xerces-init	162	77	47.53
P28	lucene-2.4	340	203	59.71	P56	zuzel	29	13	44.83

TABLE II. DESCRIPTIVE STATISTICS OF EXPERIMENTAL DATSAET CONSISTING OF 56 PROJECTS LISTED IN TABLE I

	Min	Max	Mean	Median	Std Dev	Q1	Q3
# CLS	18	23014	1470.55	231.5	4285.71	64.5	446.50
# FLC	4	2738	212.86	54	525.56	14.5	117.00
# NFLC	6	20583	1257.70	150	3774.28	42	347.50
% Faulty	2.24	92.20	28.97	22.38	20.37	13.46	41.02

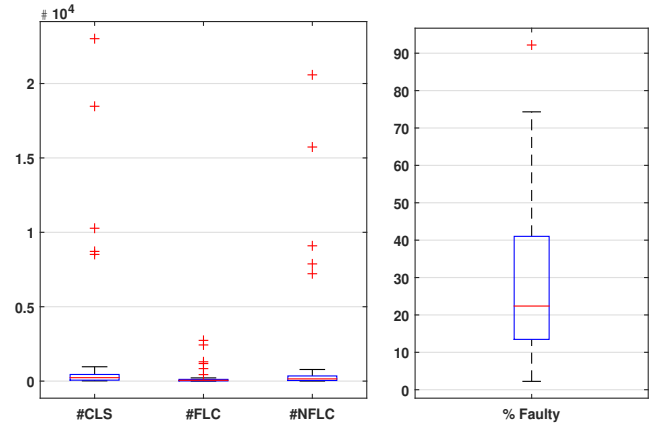

Fig. 3. Box Plot Showing the Diversity of the 56 Project Dataset Listed in Table I

experiments can be easily replicated. Table I displays our experimental dataset consisting of 56 open source projects. Table I shows the project name, number of classes, number of faulty classes and percentage of faulty classes. Our dataset consists of diverse projects from different domains. The experimental dataset consists of small projects having number of classes in the range of 18 to 100. The dataset consists of large projects having number of classes between 18000 to 24000. Similarly, there is a wide variance in the percentage of faulty classes. We have projects in our dataset for which the percentage of faulty classes is 3.83% and similarly projects which have percentage of faulty classes as 92.2%. Table II and Figure 3 shows the descriptive statistics and box plot for the number of classes, number of faulty classes, number of non-faulty classes and percentage of faulty classes in our experimental dataset. Table II and Figure 3 reveals a wide variance and diversity in-terms of the project size and quality and hence we believe that our experimental dataset removes several project specific biases and influences on the results.

IV. EXPERIMENTAL ANALYSIS AND RESULTS

A. *Fault Prediction Classifier Performance*

Table III and Figure 4 displays the descriptive statistics and box plot of the accuracy and f-measure values for the five classifiers. We apply a five-fold cross validation approach to obtain the accuracy and f-measure value of a classifier on one dataset. We use random partitioning to create training and test dataset during the five-fold cross validation process. Our experimental dataset consists of 56 projects and hence we compute the performance of each of the classifier on 56 dataset. The outcome of five-fold cross validation of one classifier on one dataset is one data-point and like this we obtain 56 data-points. Table III the minimum, maximum, mean, median, standard deviation, Q1 and Q3 for the 56 data-points for each of the classifier. Similarly, Figure 4 shows the

box-plot diagram derived from the 56 data-points for each of the classifier.

Table III and Figure 4 reveals that the mean accuracy of the decision tree based classifier is 77.87% and the mean value of the f-measure is 0.78. We observe that the mean accuracy of the decision tree classifier is the highest in comparison to all other classifiers. The mean accuracy of the ANN classifier is 76.56% which is next in performance to the decision tree classifier. The cells in Table III for the decision tree classifier is shaded in grey to highlight that it is relatively the best performing classifier. In terms of the f-measure, the performance of the ANN classifier is slightly better than the performance of the decision tree classifier. Table III and Figure 4 indicate that overall decision tree and ANN classifier are close to each-other in their performance and outperform all other classifiers. The box-plot in Figure 4 shows the quartiles as well as the variability in the data. We observe from Figure 4 that the spread or the degree of dispersion for the f-measure value for the nave bayes classifier is much more than the dispersion in f-measure values for other classifiers. Similarly, we observe a relatively higher dispersion in the accuracy value of the nave bayes classifier.

TABLE III. DESCRIPTIVE STATISTICS OF THE PERFORMANCE OF THE FIVE CLASSIFIERS IN-TERMS OF ACCURACY AND F-MEASURE

Accuracy							
	Min	**Max**	**Mean**	**Median**	**Std Dev**	**Q1**	**Q3**
LOGR	5.56	95.12	71.12	77.12	20.21	66.08	84.75
NBC	35.78	93.09	66.08	63.26	14.27	56.38	78.14
SVM	32.00	95.93	70.91	73.83	14.30	62.78	82.78
DT	56.00	97.76	77.87	77.75	11.22	67.01	88.49
ANN	55.00	96.14	76.56	78.82	11.27	67.42	85.71
F-Measure							
	Min	**Max**	**Mean**	**Median**	**Std Dev**	**Q1**	**Q3**
LOGR	0.00	0.98	0.75	0.84	0.24	0.67	0.91
NBC	0.00	0.96	0.67	0.72	0.22	0.51	0.85
SVM	0.12	0.98	0.75	0.82	0.19	0.70	0.89
DT	0.00	0.99	0.78	0.87	0.25	0.76	0.94
ANN	0.00	0.98	0.79	0.86	0.19	0.71	0.92

TABLE IV. WILCOXON SIGNED RANK TEST RESULTS FOR ACCURACY METRICS

p-value					
	LOGR	**NBC**	**SVM**	**DT**	**ANN**
LOGR	1.000	0.014	0.102	0.013	0.000
NBC	0.014	1.000	0.017	0.000	0.000
SVM	0.102	0.017	1.000	0.000	0.000
DT	0.013	0.000	0.000	1.000	0.257
ANN	0.000	0.000	0.000	0.257	1.000
Mean Difference					
	LOGR	**NBC**	**SVM**	**DT**	**ANN**
LOGR	0.000	5.041	0.213	-6.750	-5.438
NBC	-5.041	0.000	-4.828	-11.791	-10.479
SVM	-0.213	4.828	0.000	-6.963	-5.651
DT	6.750	11.791	6.963	0.000	1.312
ANN	5.438	10.479	5.651	-1.312	0.000

B. Statistical Tests on Classifier Comparison

We conduct statistical tests to investigate which of the five classifiers performs best in addition to providing accuracy and f-measure comparison data using Table III and Figure 4. Table III and Figure 4 shows the descriptive statistics of accuracy and f-measure values but do not present any information on statistical tests or significance tests. In our experiments, the null hypothesis is that there is no effect on the fault prediction accuracy due to the classification algorithm (such as decision trees, artificial neural networks, logistic regression, support vector machines and nave bayes classifier) and the classification algorithm does not make a difference. For our experiments, we initially set the p-value or the confidence threshold for the purpose of hypothesis testing as 0.05 and then adjust it to 0.005 due to Bonferroni correction [14]. According to the Bonferroni correction, significance cutoff is set at $\frac{\alpha}{n}$, where n is number of different pairs (in our case it is 5 classification techniques: n=$^{5technique}C_2 = 5*4/2 = 10$). Janez et al. discuss techniques for comparing machine learning algorithms on a single dataset as well as multiple dataset [14]. They mention that Wilcoxon test can be a more useful and a powerful technique than the paired t-test [14]. The Wilcoxon signed-rank is particularly suitable when we do a comparison of two classifiers across multiple domains. We thus perform the Wilcoxon signed-rank non-parametric test for which the results are displayed in Table IV. Table IV shows the results of the accuracy metrics. We do not show a table for the f-measure due to limited space in the paper but we obtained a similar conclusion.

We set the criteria that if the p-value is less then $\frac{0.05}{10} = 0.005$ then the null hypothesis is rejected. There are 10 comparison points as we have five classifiers. Table IV reveals that five out of the ten values are less than 0.005. Since five out of ten values are below the adjusted p-value, we infer that there is a significant different between classifier performance. Based on our results, we reject the null hypothesis and conclude that the difference between the prediction accuracy of the different classifiers is not by chance and is a function of the characteristics of the underlying algorithm of the classifier. Table IV also shows the mean difference in the accuracy of the various classifiers. We observe that the mean difference between the accuracy of DT and ANN is 1.312. DT shows the best performance. The mean difference in the accuracy of DT and NBC is highest and is 11.791.

C. Relationship between NE_{cost} and % of Faulty Modules

Figure 5 displays three graphs showing the relationship between NE_{cost} and percentage of faulty modules across three levels of testing efficiencies. The three levels of testing efficiencies are (1) Low : $\delta_u = 0.1$, $\delta_i = 0.25$, $\delta_s = 0.25$, (2) Medium : $\delta_u = 0.25$, $\delta_i = 0.45$, (3) High : $\delta_s = 0.5$ and $\delta_u = 0.5$, $\delta_i = 0.60$, $\delta_s = 0.65$ (high) where δ_u, δ_i and δ_s represents testing efficiency for unit, integration and system testing respectively as described by Kumar et al. [3][15] and Wagner et al. [8]. Figure 5 shows the NE_{cost} value for all the 56 projects in the experimental dataset. The graphs in Figure 5 uses the normalized fault removal cost and the normalized fault identification efficiency values. All the three graphs in Figure 5 shows a line for NE_{cost} = 1. We observe from Figure 5 that few points are above the NE_{cost} = 1 line and a large number of points are below the NE_{cost} = 1 line.

The number of points above the NE_{cost} = 1 line increases as the testing efficiency increases. Figure 5 shows that when the percentage of faulty classes is above 80% then the NE_{cost} value is more than 1 for all the tree levels of testing efficiency. However, when the percentage of faulty classes is below 80% then the value of the NE_{cost} can be above or below the NE_{cost} = 1 line depending on the testing efficiency and the

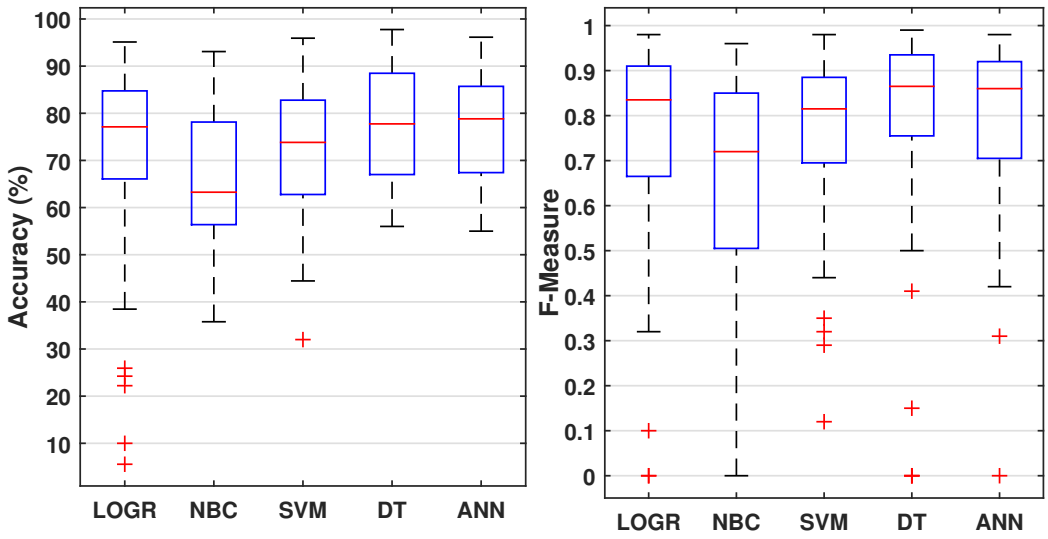

Fig. 4. Box-Plot displaying the Descriptive Statistics of the Five Different Classifier Performance Results in-terms of Accuracy & F-Measure shown in Table III

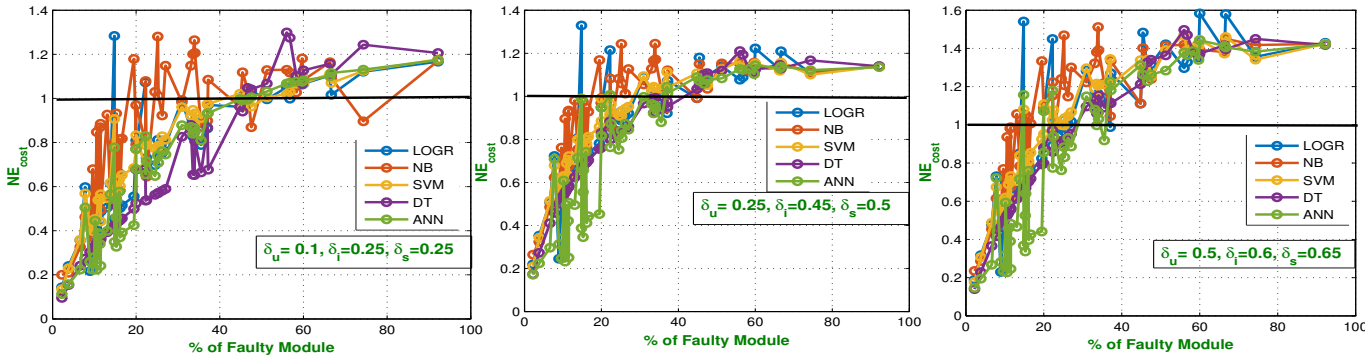

Fig. 5. Graphs displaying Relationship between NE_{cost} and Percentage of Faulty Modules for Various Levels of Testing Efficiency. Graph shows Results for Five different Classifiers

TABLE V. DERIVATION OF THRESHOLD VALUE FOR PERCENTAGE OF FAULTY CLASSES ACROSS MULTIPLE LEVELS OF TESTING EFFICIENCY

	$\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$			$\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$			$\delta_u = 0.5, \delta_i = 0.60, \delta_s = 0.65$		
	Const.	**Coeff.**	**Threshold**	**Const.**	**Coeff.**	**Threshold**	**Const.**	**Coeff.**	**Threshold**
LOGR	-5.30	0.11	49.06	-5.44	0.15	37.02	-5.30	0.23	22.97
NBC	-3.04	0.09	34.20	-4.81	0.20	24.22	-7.08	0.44	16.12
SVM	-21.27	0.46	46.01	-20.19	0.65	30.98	-7.81	0.35	22.29
DT	-2289.16	49.70	46.06	-438.82	10.64	41.23	-532.02	18.28	29.10
ANN	-3176.96	67.45	47.10	-9.76	0.27	36.61	-5.69	0.19	29.51
ALL	-5.85	0.13	44.43	-6.27	0.19	33.86	-5.46	0.23	23.90

algorithm. We observe from Figure 5 that as the percentage value of faulty classes increases, the fault-prediction technique tends to have a higher value of NE_{cost}. This means that the fault prediction can be useful for software projects with percentage of faulty classes having less than a certain threshold value. From Figure 5 we can infer that that the NE_{cost} and % of faulty classes have an association with each-other. Hence from Figure 5, we have identified an association relationship between the NE_{cost} and % of faulty classes in a software system.

D. Threshold Value Determination for % of Faulty Classes

For a given testing efficiency, we divide the NE_{cost} into two groups. One group consists of software systems for which fault prediction will be useful ($NE_{cost} < 1$). The other group consists of software system for which the fault prediction is not useful ($NE_{cost} >= 1$) for the given testing efficiency. We define a logistic regression model where the dependent variable is takes two values i.e., prediction being useful = 1 and prediction not being useful = 0. In our case the predictor variable is the percentage of faulty classes and our objective is to determine the value of the percentage of faulty classes

Fig. 6. Decision Tree Derived to Define Association between Source Code Characteristics and Fault Prediction Usefulness from a Cost Framework Perspective ($\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$)

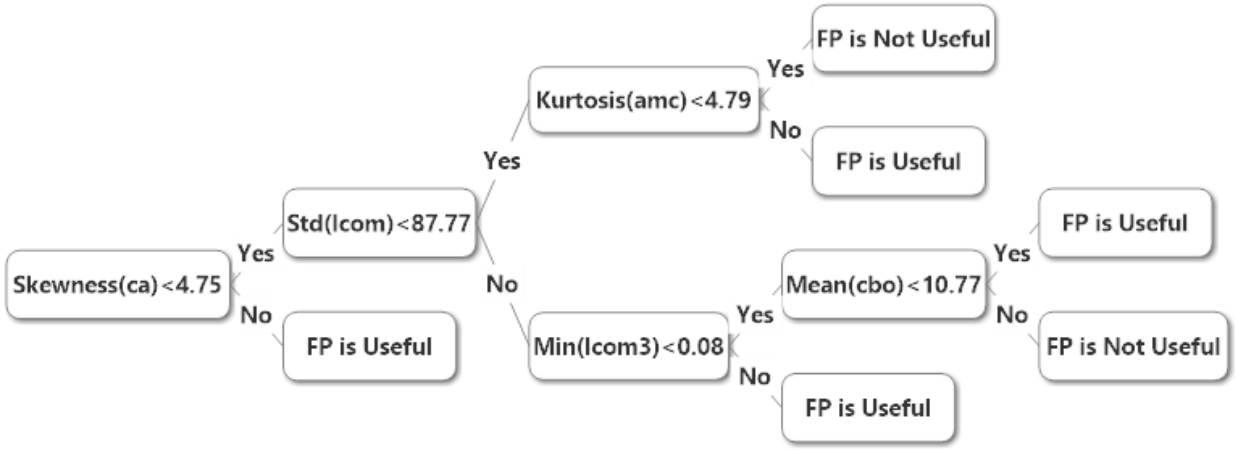

Fig. 7. Decision Tree Derived to Define Association between Source Code Characteristics and Fault Prediction Usefulness from a Cost Framework Perspective ($\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$)

TABLE VI. DISTRIBUTIONAL CHARACTERISTICS OF SOURCE CODE METRICS

	WMC			DIT			CBO			LCOM			LOC		
	Mean	**Max**	**Std Dev**	**Mean**	**Max**	**Std Dev**	**Mean**	**Max**	**Std Dev**	**Mean**	**Max**	**Std Dev**	**Mean**	**Max**	**Std Dev**
ant-1.3	10.59	71.00	10.36	2.28	6.00	1.28	10.43	103.00	14.89	69.32	2247.00	259.28	301.59	2193.00	339.11
ivy-1.1	10.61	82.00	11.63	1.67	6.00	1.22	10.42	72.00	9.74	87.08	3021.00	335.95	245.87	3982.00	486.39
lucene-2.4	10.39	166.00	13.44	1.81	5.00	0.89	10.75	128.00	12.16	68.72	6747.00	442.98	302.53	8474.00	639.33
synapse-1.1	7.33	61.00	8.28	1.60	5.00	0.80	12.50	82.00	10.87	34.81	1172.00	130.75	190.55	1164.00	200.61
zuzel	13.93	47.00	10.89	2.97	6.00	2.44	7.48	42.00	8.44	85.10	707.00	144.48	497.28	1910.00	619.42

such that the dependent variable changes. We apply logistic regression and determine the constant, coefficient and the threshold value using a logistic function. Table V shows the constant, coefficient, and threshold value in terms of the % of faulty classes for different values of δ_u, δ_i and δ_s. Our analysis reveals that the threshold value for medium testing efficiency is lowest for the NBC classifier (24.22%) and highest for the decision tree classifier (41.23%). Table V reveals that the threshold value for ANN is highest for high testing efficiency and slightly more than the threshold value for DT. Thus, in addition to the results presented in Figure 5, we further establish the relationship between the percentage of faulty classes and usefulness from a cost perspective. From Table V, we observe that the fault prediction models are suitable for projects with faulty classes less than the threshold value depending on fault identification efficiency (low- 44.43%, medium- 33.86% and high- 23.90%).

TABLE VII. DECISION TREE CLASSIFIER PERFORMANCE FOR THE TASK OF FAULT PREDICTION USEFULNESS (FM = F-MEASURE)

	Accuracy	**Precision**	**Recall**	**FM**
$\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$	96.00	1.00	0.86	0.92
$\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$	100.00	1.00	1.00	1.00
$\delta_u = 0.5, \delta_i = 0.60, \delta_s = 0.65$	91.00	0.81	1.00	0.90

E. Decision Tree Induction for Fault Prediction Usefulness

We conduct experiments based on the process defined in the architecture diagram illustrated in Figure 2. For every level of the testing efficiency (less, medium and high) and for each of the 56 projects, we compute the value of the target variable which is a binary variable denoting whether fault proneness is useful or not. Thus we have an annotated dataset for building a classifier. The number of input features are 200 (20 source code metrics and 10 statistical characteristics for each metric) and using this we create three decision trees for each of the testing efficiency.

Table VI displays 3 distributional characteristics of 4 out

of 20 source code metrics for 5 projects in our dataset. We show a subset of the results due to limited space in the paper. The projects selected for the analysis presented in Table VI are such that they represent diversity. For example, the percentage of faulty modules in ant-1.3 is 37.04% whereas the percentage of faulty modules in lucene-2.4 is 59.71%. Table VI reveals that there is a wide variance in the distributional characteristics of source code metrics across projects. For example, the maximum WMC value of lucene-2.4 is more than 3 times of the maximum WMC value of zuzel. The standard deviation of CBO is 14.89 for the ant-1.3 project which is much higher than the standard deviation of ivy-1.1 and zuzel.

Figure 6 and 7 shows the decision trees for two of the testing efficiency levels. We do not present the third decision tree due to limited space in the paper. Our research objective is to investigate whether the usefulness of fault prediction results using a cost evaluation model are related to distributional characteristics of source code metrics. We apply a decision tree based approach on the generated source code distributional characteristics and fault prediction usefulness result instances because decision tree resulted in the best performance for fault prediction in our experiments and also the results of the decision tree are more easy to interpret. Our test is that if the decision tree can with high accuracy and purity can be constructed then the characteristics of source code metrics are related to cost evaluation framework based fault prediction usefulness results. We consider a five-fold cross-validation approach to compute the accuracy of the decision tree created by us.

Figure 6, Figure 7 and Table VII shows that a short decision tree with high accuracy and purity can be created and hence the source code distributional characteristics are good predictors of whether a fault prediction model is useful or not for a given testing efficiency level and percentage of faulty modules. Figure 6 shows that the skewness of Ca (afferent couplings) and WMC (weighted methods per class) is an indicator for determining fault proneness usefulness. Similarly, Figure 7 reveals that the standard deviation value of LCOM (lack of cohesion in methods) and the kurtosis value of AMC (average method complexity) are also indicators for the target class. Table VII shows the performance of the decision tree classifier in terms of its effectiveness in determining whether fault prediction is useful from a cost perspective based on the source code metrics. Table VII shows that the accuracy, precision, recall and f-measure of the decision tree is high. The accuracy varies from 91% to 100%. The f-measure value varies from 0.90 to 1.00.

V. Conclusion

We build fault prediction models on 56 open-source projects using 5 different classification algorithms. We conclude that decision tree classifier performs best in-terms of average accuracy and applying five-fold cross validation approach. ANN classifier performs best in-terms of mean f-measure. We conduct statistical tests to accept or reject the null hypothesis and to determine if the accuracy is a function of the classifier characteristics. We apply Wilcoxon test using Bonferroni correction in our analysis for comparing the 5 algorithms on 56 dataset. We reject the null hypothesis and infer that the difference between the prediction accuracy of ANN, DT, SVM, LOGR and NBC classifiers is not by chance and is a function of the characteristics of the classification technique. We compute the usefulness of the fault prediction approach from a cost perspective as a function of the percentage of faulty modules in the software system across different levels of testing efficiency. We demonstrate that there is a relationship between the percentage of faulty modules and fault prediction usefulness for a given level of testing efficiency. We compute several distributional characteristics of source code metrics and show that there is a relationship between the distributional characteristics of the source code metrics and fault prediction usefulness. We derive decision trees which can classify whether faulty prediction is useful or not from a cost evaluation framework based on the distributional characteristics of source code across different testing efficiency.

References

[1] C. Catal, "Software fault prediction: A literature review and current trends," *Expert systems with applications*, vol. 38, no. 4, pp. 4626–4636, 2011.

[2] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.

[3] L. Kumar, S. K. Sripada, A. Sureka, and S. K. Rath, "Effective fault prediction model developed using least square support vector machine (lssvm)," *Journal of Systems and Software*, 2017.

[4] L. Kumar, S. K. Rath, and A. Sureka, "Empirical analysis on effectiveness of source code metrics for predicting change-proneness." in *ISEC*, 2017, pp. 4–14.

[5] L. Kumar and A. Sureka, "Using structured text source code metrics and artificial neural networks to predict change proneness at code tab and program organization level." in *ISEC*, 2017, pp. 172–180.

[6] Y. Jiang and B. Cukic, "Misclassification cost-sensitive fault prediction models," in *Proceedings of the 5th international conference on predictor models in software engineering*. ACM, 2009, p. 20.

[7] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 561–595, 2008.

[8] S. Wagner, "A literature survey of the quality economics of defect-detection techniques," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006, pp. 194–203.

[9] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, June 1994.

[10] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, pp. 69–81, 2010.

[11] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[12] V. U. Challagulla, F. B. Bastani, I.-L. Yen, and R. A. Paul, "Empirical assessment of machine learning based software defect prediction techniques," in *Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005. 10th IEEE International Workshop on*. IEEE, 2005, pp. 263–270.

[13] "The promise repository of empirical software engineering data," 2015.

[14] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[15] L. Kumar and S. K. Rath, "Empirical validation for effectiveness of fault prediction technique based on cost analysis framework," *International Journal of System Assurance Engineering and Management*, pp. 1–14, 2016.