Analyzing Fault Prediction Usefulness from Cost Perspective using Source Code Metrics

Lov Kumar[1] Ashish Sureka[2]

[1]NIT Rourkela, India (lovkumar505@gmail.com)

[2]ABB India, India (ashish.sureka@in.abb.com)

IC3 2017

Table of Contents

1. Research Motivation and Aim
   - Objectives and Context Setting
2. Research Contributions
3. Research Framework and Solution Approach
4. Experimental Dataset
5. Experimental Results
6. Conclusion
7. References

Table of Contents

1 Research Motivation and Aim
- Objectives and Context Setting

2 Research Contributions

3 Research Framework and Solution Approach

4 Experimental Dataset

5 Experimental Results

6 Conclusion

7 References

Software Fault Prediction

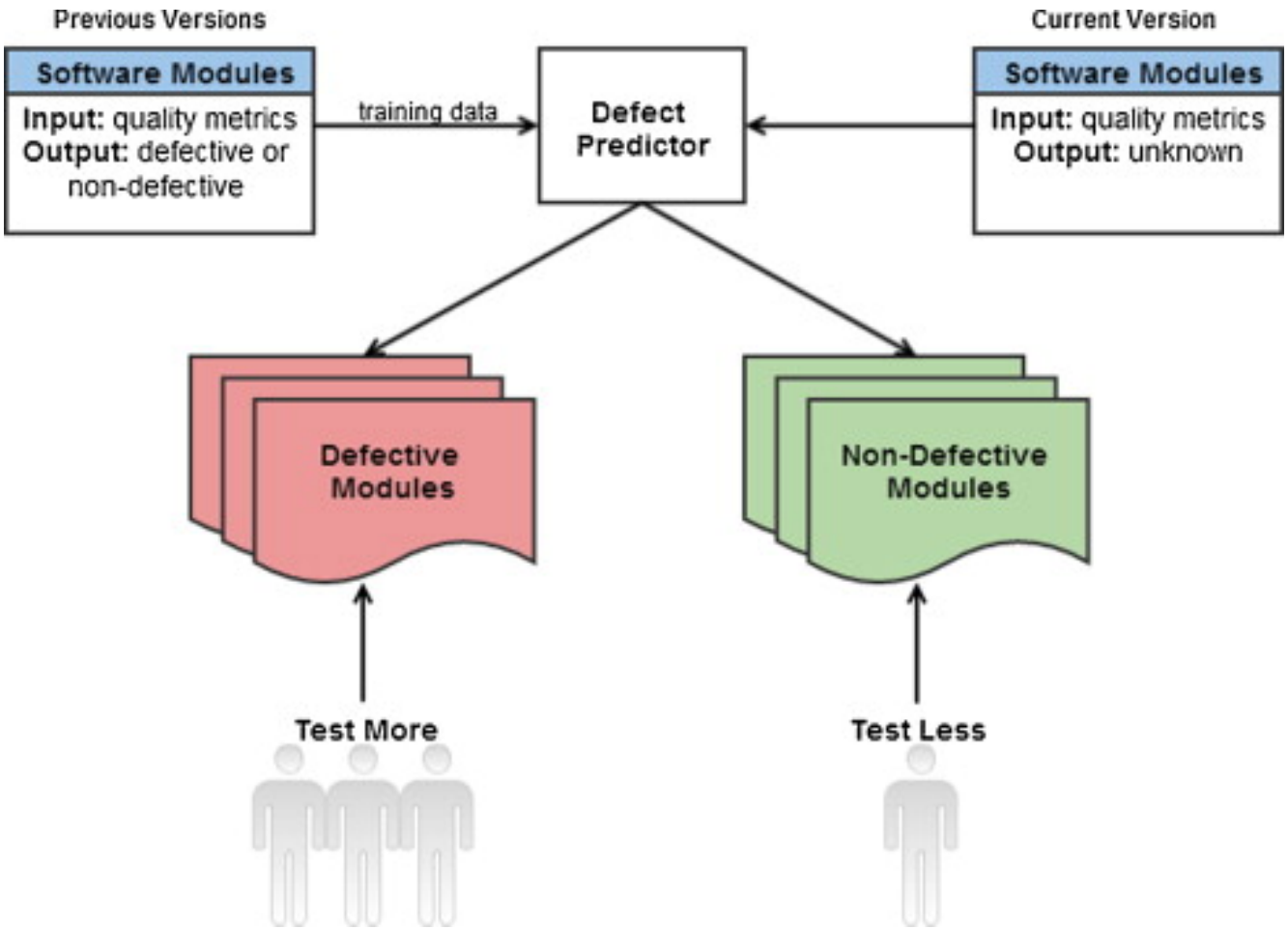
Statistical model to estimate the fault-proneness

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 [2][7][13]

Useful for estimating the quality of a software and optimizing test resource allocation

Several studies are conducted on the application of source code metrics and machine learning algorithms for fault-proneness prediction [2][7][13]

Software Fault Prediction Approach



Software Fault Prediction

Cost and Economics

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 [13][8][9][16]

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 [8][9]

Wagner et al. discuss return on investment for a fault-detection method [16]

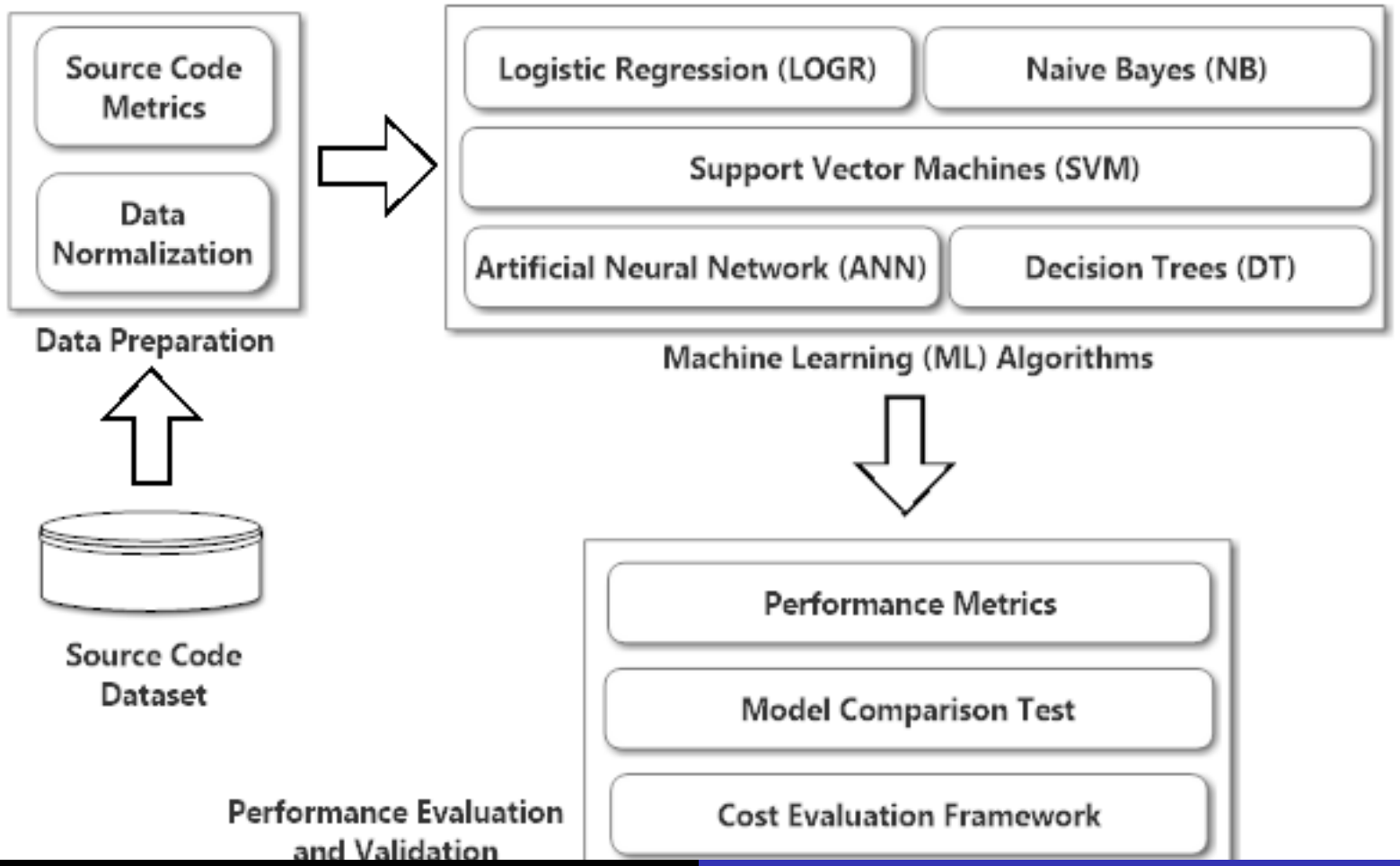
Novel and Unique Contributions

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

We propose a novel approach to predict fault proneness using the distributional characteristics of source code

Architecture Diagram - Fault Prediction Model



Architecture Diagram - Multi-Step Process

We apply 5 different machine learning techniques to build fault-prediction models

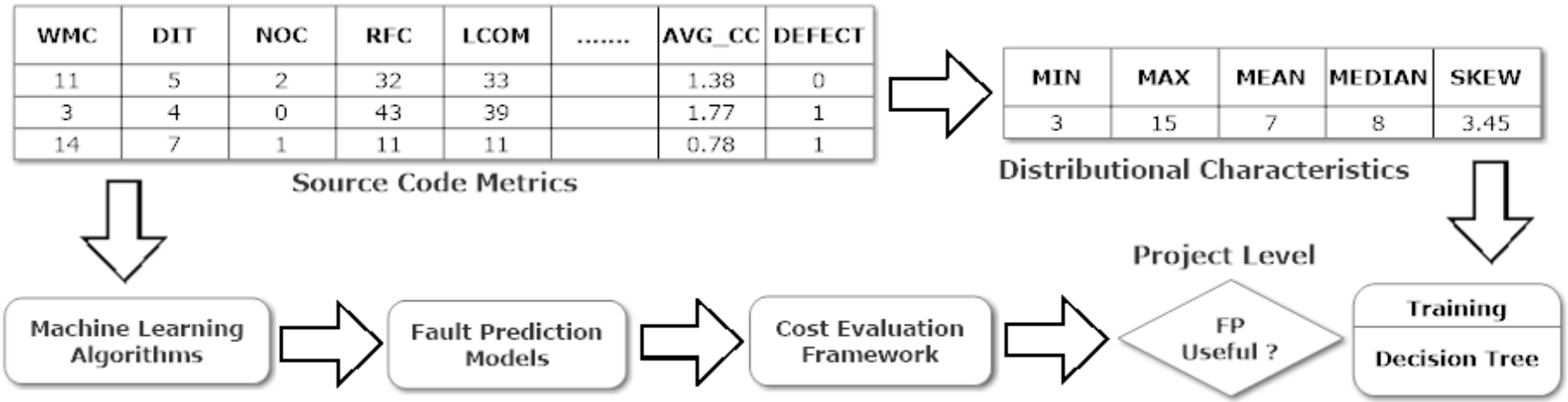
We use dataset from 56 projects

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

We use the cost evaluation framework defined by Kumar et al. and Wagner et al. [13][16]

Model Using Distributional Characteristics of Source Code



Architecture Diagram - Multi-Step Process

Fault prediction usefulness model incorporating cost factors from distributional characteristics of source code metrics

Our approach is a multi-step process consist of creating an annotated data of fault prediction usefulness for 56 projects

We then build a decision tree classifier using distribution characteristics of source code metrics as attributes of the tree.

Table: 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

Experimental Dataset - from PROMISE Repository

Table: 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
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

Dataset Details

We conduct experiments on dataset downloaded from the PROMISE[a] repository [1]

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

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

Descriptive Statistics of Experimental Datsaet

	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

Descriptive statistics for the number of classes, number of faulty classes, number of non-faulty classes and percentage of faulty classes in our experimental dataset

Descriptive Statistics of Dataset

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%

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

Performance of the Five Classifiers - Accuracy

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

Performance of the Five Classifiers - F-Measure

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

Insights from Experiments - I

We apply a five-fold cross validation approach to obtain the accuracy and f-measure value of a classifier on one dataset

Mean accuracy of the decision tree based classifier is 77.87% and the mean value of the f-measure is 0.78

Mean accuracy of the decision tree classifier is the highest in comparison to all other classifiers

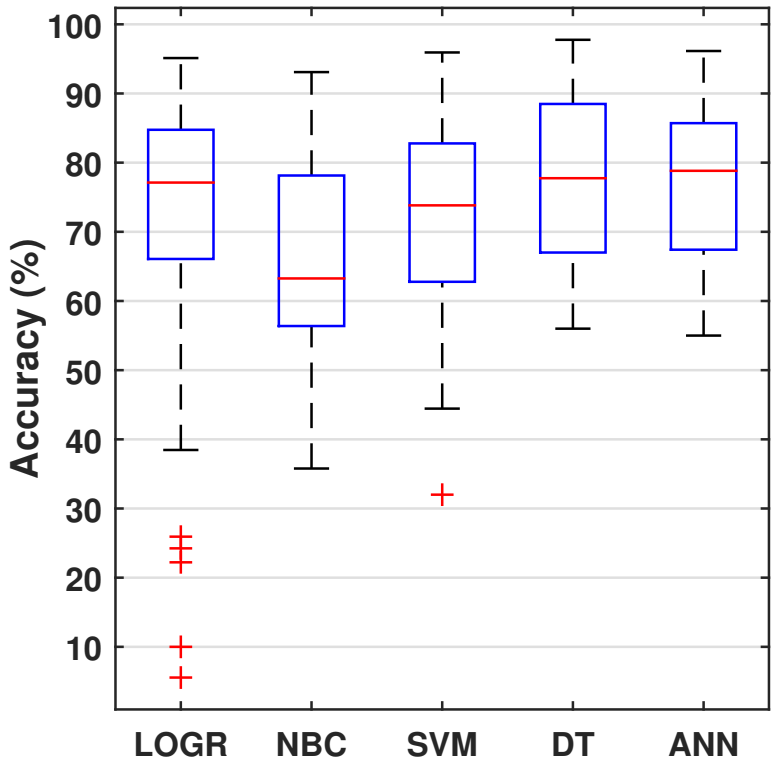
Insights from Experiments - II

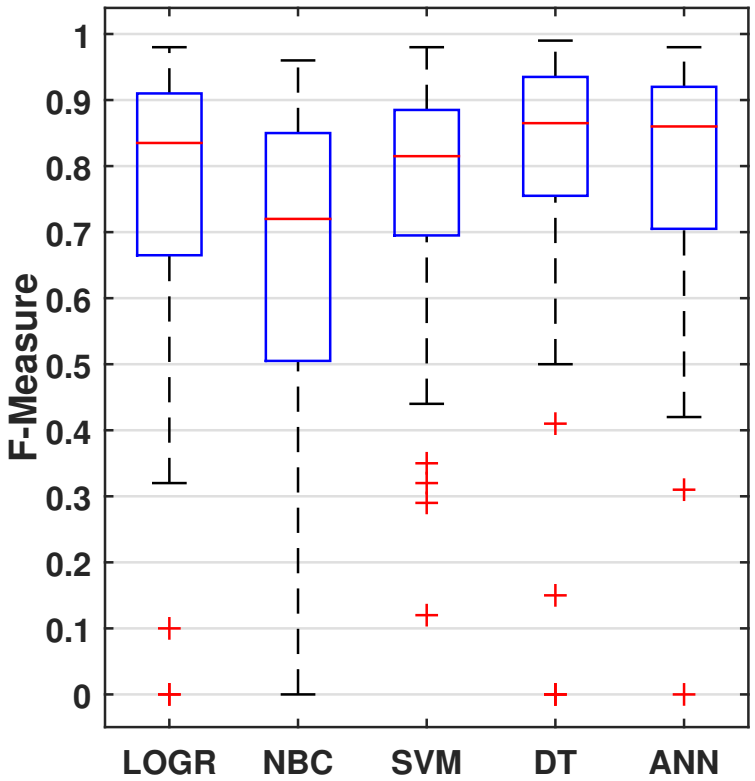
Mean accuracy of the ANN classifier is 76.56% which is next in performance to the decision tree classifier

In terms of the f-measure, the performance of the ANN classifier is slightly better than the performance of the decision tree classifier

Overall decision tree and ANN classifier are close to each-other in their performance and outperform all other classifiers

Five Different Classifier Performance Result





Insights from Experiments - III

The box-plot shows the quartiles as well as the variability in the data

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

Relatively higher dispersion in the accuracy value of the nave bayes classifier

Wilcoxon Signed Rank Test - Accuracy

Table: 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

Wilcoxon Signed Rank Test - Accuracy

Table: Wilcoxon Signed Rank Test Results for Accuracy Metrics

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

Statistical Tests on Classifier Comparison - I

The null hypothesis is that there is no effect on the fault prediction accuracy due to the classification algorithm

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 [5]

The Wilcoxon signed-rank is particularly suitable when we do a comparison of two classifiers across multiple domains

Statistical Tests on Classifier Comparison - II

We set the criteria that if the p-value is less then $\frac{0.05}{10} = 0.005$ then the null hypothesis is rejected

Five out of ten values are below the adjusted p-value, we infer that there is a significant different between classifier performance

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.

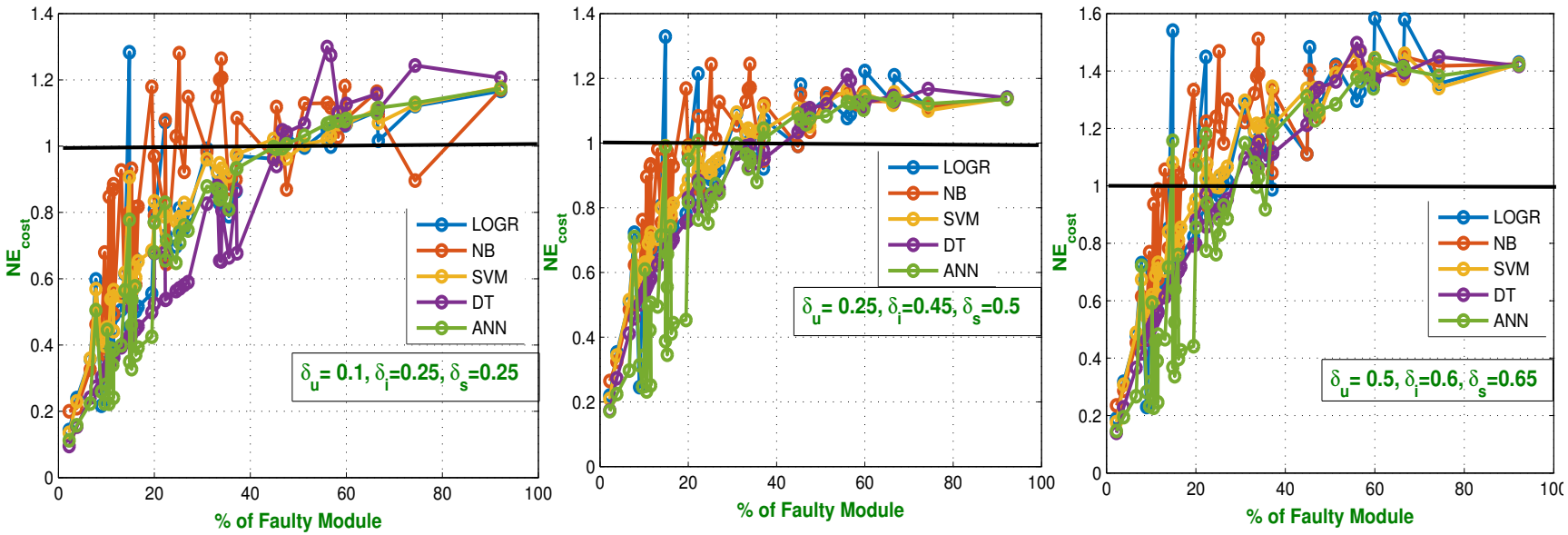


Figure: Graphs displaying Relationship between NE_{cost} and Percentage of Faulty Modules for Various Levels of Testing Efficiency. Graph shows Results for Five different Classifiers

Relationship between NE_{cost} and % of Faulty Modules - I

All the three graphs shows a line for $NE_{cost} = 1$. We observe from Figure 2 that few points are above the $NE_{cost} = 1$ line and a large number of points are below the $NE_{cost} = 1$ line

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

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 algorithm

Relationship between NE_{cost} and % of Faulty Modules - II

The percentage value of faulty classes increases, the fault-prediction technique tends to have a higher value of NE_{cost}

Fault prediction can be useful for software projects with percentage of faulty classes having less than a certain threshold value

We have identified an association relationship between the NE_{cost} and % of faulty classes in a software system

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

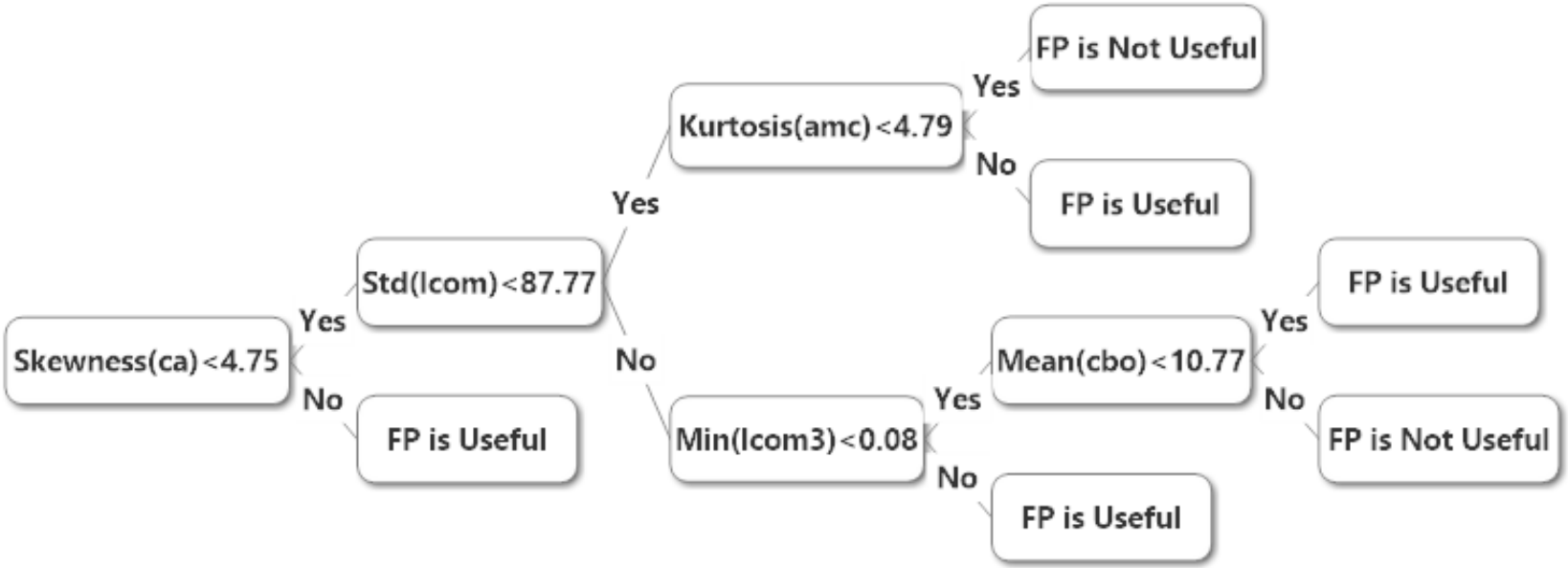


Figure: Decision Tree Derived to Define Association between Source Code Characteristics and Fault Prediction Usefulness from a Cost Framework Perspective ($\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$)

Decision Tree Induction for Fault Prediction Usefulness - I

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

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

Decision Tree Induction for Fault Prediction Usefulness - II

Figure in previous slides shows the decision trees for two of the testing efficiency levels

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

A short decision tree with high accuracy and purity can be created

Decision Tree Classifier Performance

Table: 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

Decision Tree Classifier Performance

Table in previous slide 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

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

Conclusion - I

We conclude that decision tree classifier performs best in-terms of average accuracy and applying five-fold cross validation approach

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 demonstrate that there is a relationship between the percentage of faulty modules and fault prediction usefulness for a given level of testing efficiency

Conclusion II

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 I

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

[2] Cagatay Catal. Software fault prediction: A literature review and current trends. *Expert systems with applications*, 38(4):4626–4636, 2011.

[3] Venkata UB Challagulla, Farokh B Bastani, I-Ling Yen, and Raymond 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*, pages 263–270. IEEE, 2005.

[4] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[5] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[6] Iker Gondra. Applying machine learning to software fault-proneness prediction. *Journal of Systems and Software*, 81(2):186–195, 2008.

References II

[7] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2012.

[8] Yue Jiang and Bojan Cukic. Misclassification cost-sensitive fault prediction models. In *Proceedings of the 5th international conference on predictor models in software engineering*, page 20. ACM, 2009.

[9] Yue Jiang, Bojan Cukic, and Yan Ma. Techniques for evaluating fault prediction models. *Empirical Software Engineering*, 13(5):561–595, 2008.

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

[11] Lov Kumar and Santanu Ku Rath. Empirical validation for effectiveness of fault prediction technique based on cost analysis framework. *International Journal of System Assurance Engineering and Management*, pages 1–14, 2016.

References III

[12] Lov Kumar, Santanu Kumar Rath, and Ashish Sureka. Empirical analysis on effectiveness of source code metrics for predicting change-proneness. In *ISEC*, pages 4–14, 2017.

[13] Lov Kumar, Sai Krishna Sripada, Ashish Sureka, and Santanu Ku. Rath. Effective fault prediction model developed using least square support vector machine (lssvm). *Journal of Systems and Software*, 2017.

[14] Lov Kumar and Ashish Sureka. Using structured text source code metrics and artificial neural networks to predict change proneness at code tab and program organization level. In *ISEC*, pages 172–180, 2017.

[15] Ruchika Malhotra. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27:504–518, 2015.

[16] Stefan 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*, pages 194–203. ACM, 2006.