An Empirical Analysis on Effective Fault Prediction Model Developed using Ensemble Methods

Lov Kumar[1] Santanu Rath[1] Ashish Sureka[2]

[1]NIT Rourkela, India
Email: lovkumar505@gmail.com, skrath@nitrkl.ac.in

[2]ABB India, India
Email: ashish.sureka@in.abb.com

COMPSAC 2017

Table of Contents

Table of Contents

Fault or Defect Prediction

Why Identify Fault-Prone Classes ?

Early identification of source code regions, classes or modules where faults are likely to occur.

Helps in optimizing and guiding testing efforts and hence results in improvement of software quality

Hall et al. [14] and Catal et al. [7] conduct a systematic literature review in the area of fault prediction.

Arisholm et al. conduct a examination of methods to develop and evaluate the performance of fault prediction models [1].

Research Gaps and Focus

Cost Evaluation Model and Ensemble Methods

Application of base learners: logistic regression, ANN and radial basis function neural network as constituents of ensemble methods

Generalizability: experiments conducted on dataset belonging to **45** open source projects

Propose a cost evaluation model which takes into account the economics of software quality assurance [21].

Normalized fault removal cost in the fault prediction framework based on ensemble methods is a novel contribution [1].

Table of Contents

Dependent and Independent Variables

Dependent Variable	**Independent Variable**
Fault	DIT, WMC, RFC, CBO, LCOM, NOC, Ce, Ca, LCOM3, NPM, DAM, MOA, LOC, CAM, MFA, CBM, AVG-CC, MAX-CC, AMC, IC
Fault	Reduced feature attributes using proposed framework

Source code metrics is input variables or predictors. The class or category bugs is a dependent variable

Table of Contents

List of 45 Open Source Projects (Part 1)

Id	Project	No. of class	No. of Faulty class	Faulty (%)
D1	ant-1.3	125	20	16
D2	ant-1.4	178	40	22.47
D3	ant-1.5	293	32	10.92
D4	ant-1.6	351	92	26.21
D5	ant-1.7	745	166	22.28
D6	arc	234	27	11.54
D7	berek	43	16	37.21
D8	camel-1.0	339	13	3.83
D9	camel-1.2	608	216	35.53
D10	camel-1.4	872	145	16.63
D11	camel-1.6	965	188	19.48
D12	e-learning	64	5	7.81
D13	ivy-1.1	111	63	56.76
D14	ivy-1.4	241	16	6.64
D15	ivy-2.0	352	40	11.36
D16	jedit-3.2	272	90	33.09
D17	jedit-4.0	306	75	24.51

List of 45 Open Source Projects (Part 2)

Id	Project	No. of class	No. of Faulty class	Faulty (%)
D18	jedit-4.1	312	79	25.32
D19	jedit-4.2	367	48	13.08
D20	kalkulator	27	6	22.22
D21	log4j-1.0	135	34	25.19
D22	log4j-1.1	109	37	33.94
D23	log4j-1.2	205	189	92.2
D24	lucene-2.0	195	91	46.67
D25	lucene-2.2	247	144	58.3
D26	lucene-2.4	340	203	59.71
D27	pdftranslator	33	15	45.45
D28	prop-1	18471	2738	14.82
D29	prop-2	23014	2431	10.56
D30	prop-3	10274	1180	11.49
D31	prop-4	8718	840	9.64
D32	prop-5	8516	1299	15.25
D33	prop-6	660	66	10
D34	redaktor	176	27	15.34

List of 45 Open Source Projects (Part 3)

Id	Project	No. of class	No. of Faulty class	Faulty (%)
D35	serapion	45	9	20
D36	synapse-1.0	157	16	10.19
D37	synapse-1.1	222	60	27.03
D38	synapse-1.2	256	86	33.59
D39	termoproject	42	13	30.95
D40	velocity-1.5	214	142	66.36
D41	velocity-1.6	229	78	34.06
D42	xerces-1.2	440	71	16.14
D43	xerces-1.3	453	69	15.23
D44	xerces-1.4	588	437	74.32
D45	xerces-init	162	77	47.53

45 real-world datasets from the PROMISE repository [a].

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

Table of Contents

Investigating - Source Code Metrics

[RQ1] Do source code metrics predict faulty or non faulty classes?

[RQ2] Can the selected set of source code metrics better predict whether a class is faulty or not?

[RQ3] Which fault prediction technique is a most suitable one for this purpose among all?

[RQ4] For any given software product, is performing fault prediction analyses economically effective?

Validation of Source Code Metrics

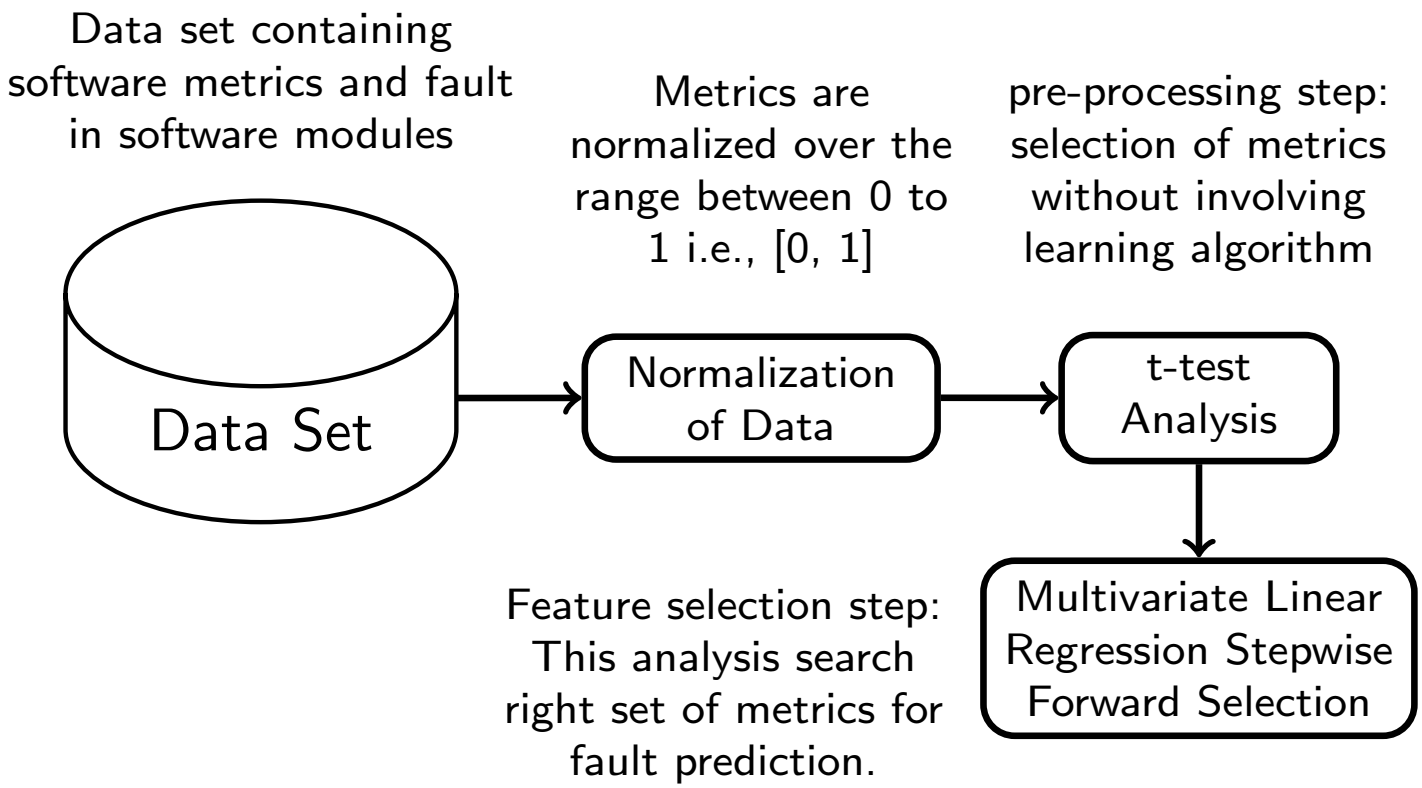
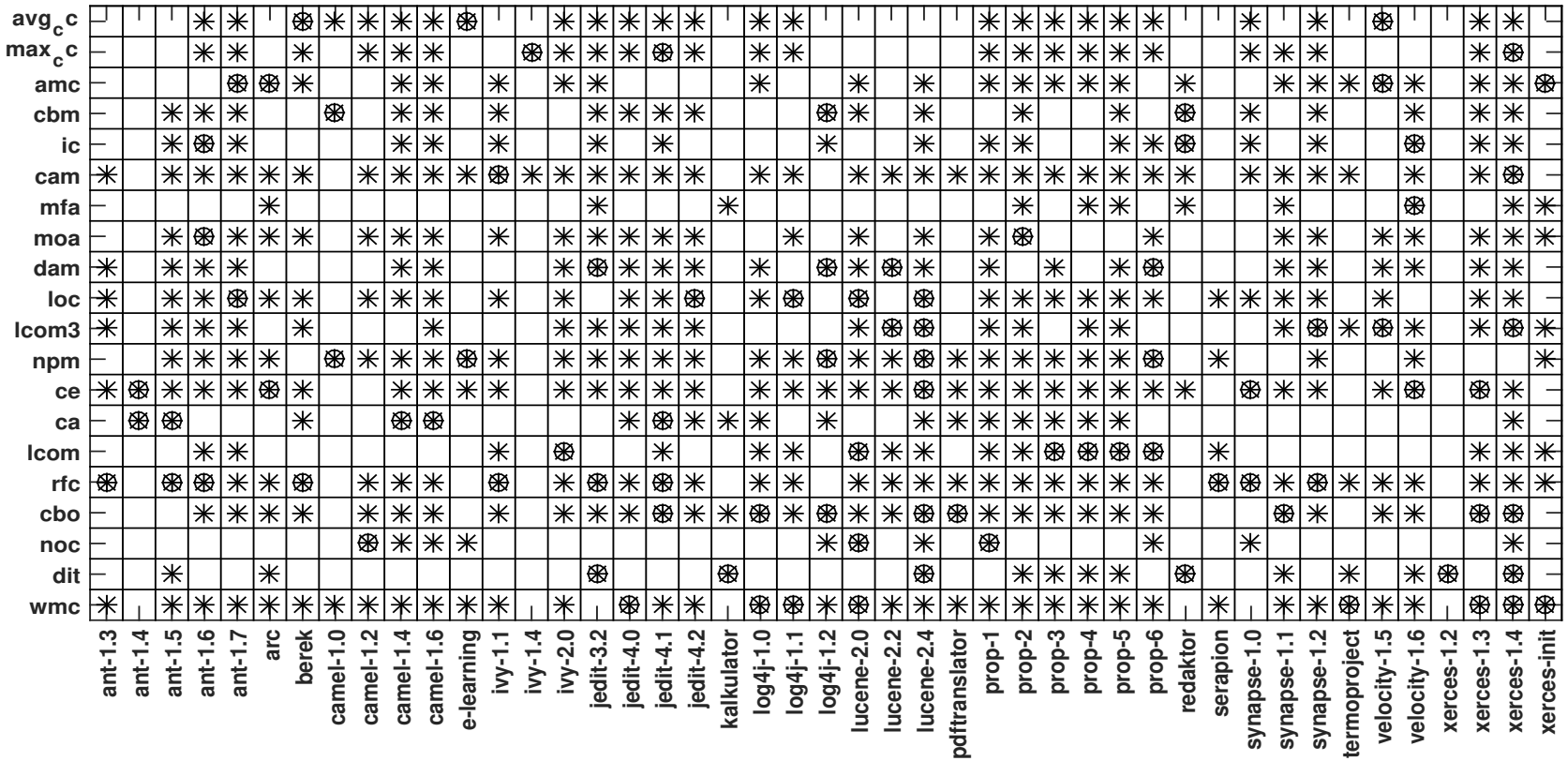


Figure: Proposed Framework of Software Metrics Validation

Selected Source Code Metrics



Selected Source Code Metrics

- We started the statistical analysis with 20 source code object-oriented metrics.
- For the purpose of simplicity, the graphs are represented using four different symbols as described below:
  - Empty circle (○): source code metrics selected after t-test analysis; and
  - Circle with star (⊛): source code metrics selected after t-test and MLR stepwise forward selection method.

Cost Analysis Model

- The cost evaluation model accounts for the realistic costs incurred to remove a fault or defect in the system based on the ideas proposed by Wagner et al. ([21]).
- **Normalized fault removal cost (NE_{cost})** : Normalized fault removal cost ($\frac{E_{cost}}{T_{cost}}$) and its interpretation can be modeled as:

$$\text{If the value of } NEcost = \begin{cases} < 1, \text{then application of fault prediction is useful} \\ => 1, \text{then application of testing methodlogies may be helpful} \end{cases} \quad (1)$$

Experimental setup

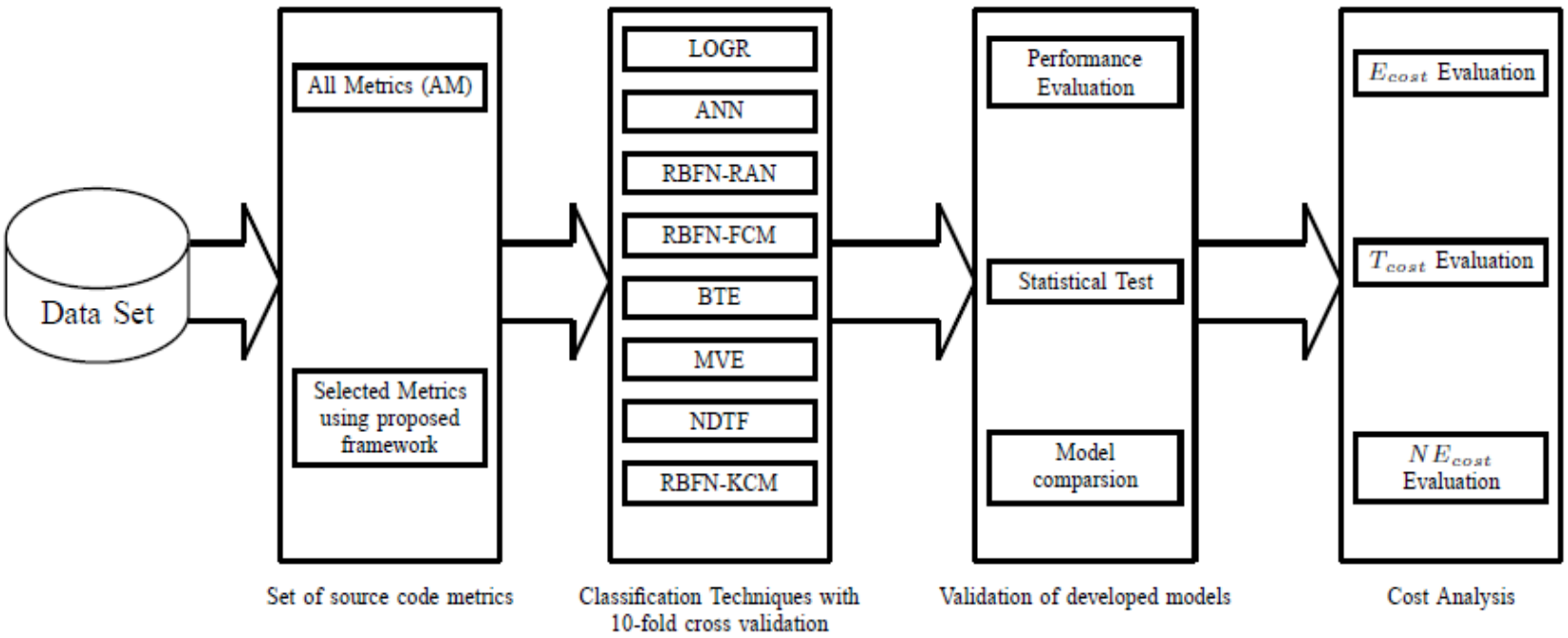


Figure: Framework of Proposed work

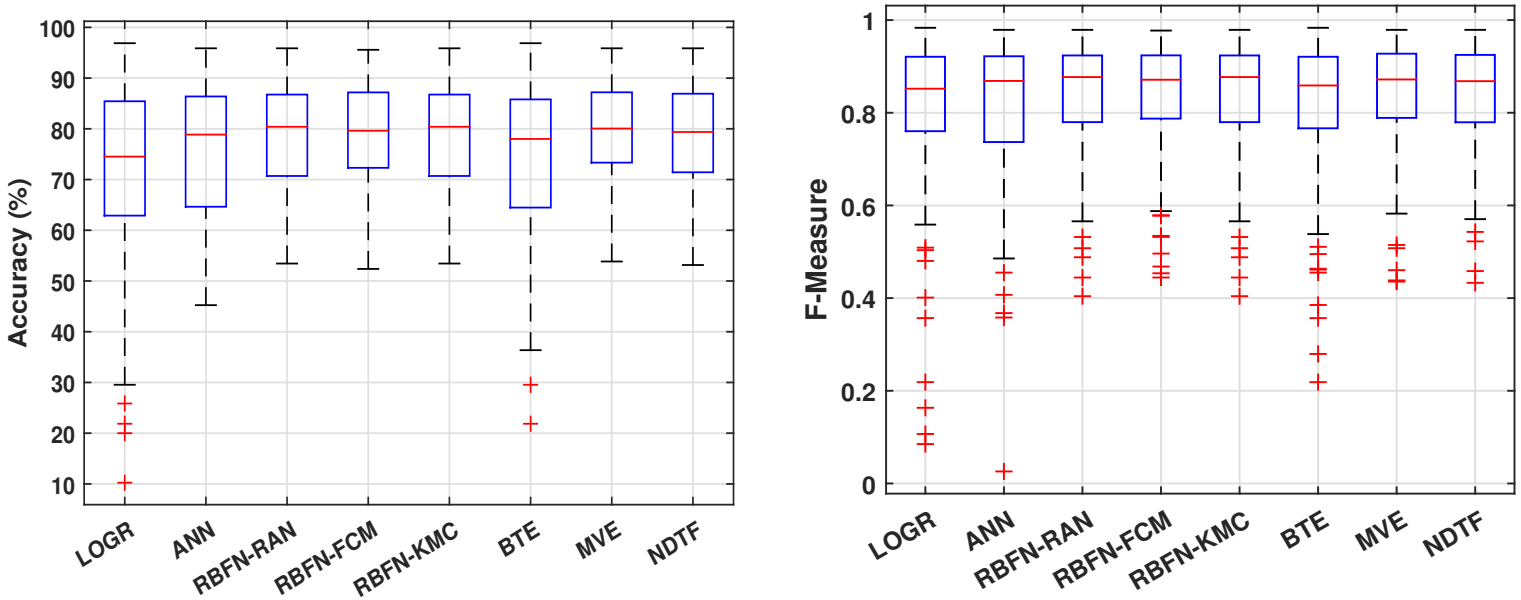
Experimental setup

- **Metrics Selection:** Selection of suitable set of source code metrics using proposed source code metrics validation framework.
- **Prediction Model:** Development of prediction model by considering source code metrics as input to predict fault proneness.
- **Performance Measures:** Selection of performance measures that can used to evaluate the predictive capability of fault prediction models.

Experimental setup

- **Validation Methods:** Use of efficient validation methods to determine the true predictive applicability of the developed models.
- **Statistical Tests:** Selection of appropriate statistical tests to determine the superiority of one prediction technique over the other prediction techniques and also determine the superiority of one set of source code metrics over the other sets.
- **Experimental Validation:** Validation of developed fault prediction models using proposed cost analysis framework.

Classification Techniques

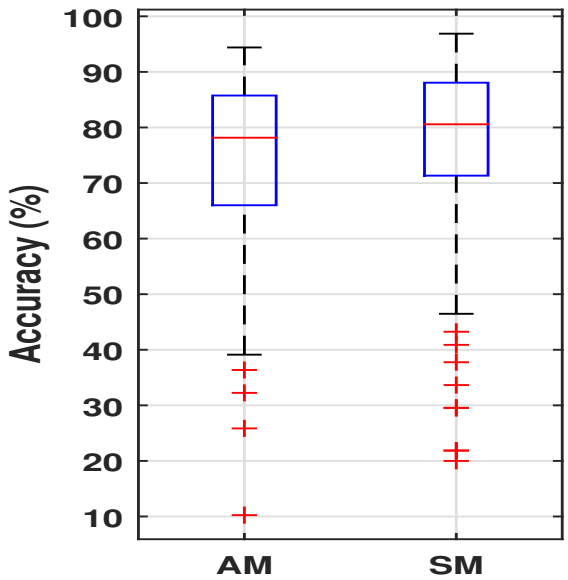


((a)) Accuracy

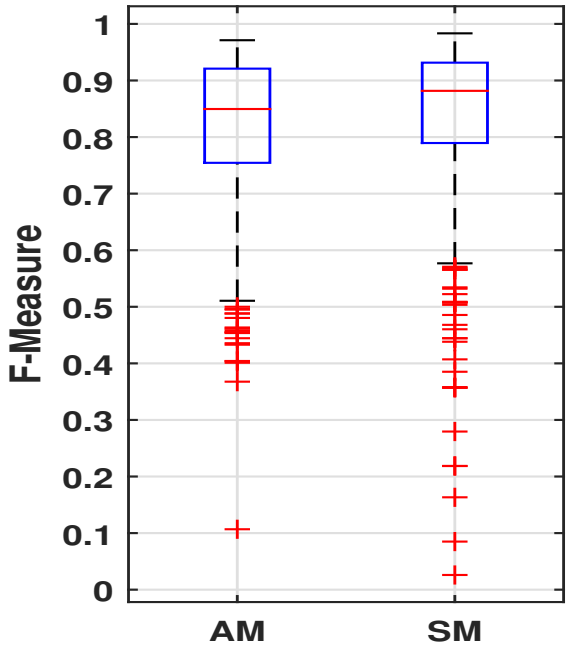
((b)) F-Measure

Figure: Classification Techniques

Selected Set of Source Code Metrics



((a)) Accuracy



((b)) F-Measure

Experimental results

From the box-plot diagram, it can be inferred that:

- In all cases, the selected set of source code metrics has a high median value. Based on the box-plots, SM produced the best result, i.e. the proposed software metrics validation method computes the best set of source code metrics for predicting faulty and non-faulty classes of object-oriented software as compared to all metrics.
- Among all classification, ensemble methods have outperformed as compared to individual models. Further, It is observed that MVE yields better results compared to other techniques.

Table of Contents

Classification Techniques

Table: t-test

Accuracy								
Mean								
	LOGR	**ANN**	**RBFN-RAN**	**RBFN-FCM**	**RBFN-KCM**	**BTE**	**MVE**	**NDTF**
LOGR	0.00	-6.46	-8.78	-8.85	-8.78	-2.98	-9.18	-8.86
ANN	6.46	0.00	-2.32	-2.40	-2.32	3.48	-2.72	-2.41
RBFN-RAN	8.78	2.32	0.00	-0.08	0.00	5.80	-0.40	-0.09
RBFN-FCM	8.85	2.40	0.08	0.00	0.08	5.87	-0.33	-0.01
RBFN-KCM	8.78	2.32	0.00	-0.08	0.00	5.80	-0.40	-0.09
BTE	2.98	-3.48	-5.80	-5.87	-5.80	0.00	-6.20	-5.88
MVE	9.18	2.72	0.40	0.33	0.40	6.20	0.00	0.31
NDTF	8.86	2.41	0.09	0.01	0.09	5.88	-0.31	0.00
p-value								
	LOGR	**ANN**	**RBFN-RAN**	**RBFN-FCM**	**RBFN-KCM**	**BTE**	**MVE**	**NDTF**
LOGR	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
ANN	0.00	1.00	0.00	0.00	0.00	0.24	0.00	0.00
RBFN-RAN	0.00	0.00	1.00	0.17	1.00	0.00	0.00	0.93
RBFN-FCM	0.00	0.00	0.17	1.00	0.17	0.00	0.26	0.12
RBFN-KCM	0.00	0.00	1.00	0.17	1.00	0.00	0.00	0.93
BTE	0.00	0.24	0.00	0.00	0.00	1.00	0.00	0.00
MVE	0.00	0.00	0.00	0.26	0.00	0.00	1.00	0.02
NDTF	0.00	0.00	0.93	0.12	0.93	0.00	0.02	1.00

Classification Techniques

- Five different classification techniques and three different ensemble methods have been considered to develop a model to predict whether the class is faulty or not.
- Two different sets of metrics, one containing all metrics and one selected set using the proposed metrics validation techniques, have been considered as the input to develop fault prediction models over 45 different projects with two different performance parameters, i.e. accuracy, and F-Measure.
- Each technique a total number of two sets (one for each performance) is used, each with 90 data points ((1 feature selection method + 1 considering all features) * 45 datasets)).

Classification Techniques

- The Bonferroni correction sets the significance cutoff at $\frac{\alpha}{n}$. In this study, eight different techniques have been considered for analysis, i.e. total number of twenty eight (28) different pairs are possible ($^{8 technique}C_2 = 8 * 7/2 = 28$) and all results are analyzed at a 0.05 significance level.
- Most of the cases there is a significant difference between these approaches.The ensemble methods have outperformed when compared to individual models.

Classification Techniques

Table: t-test: All metrics and Selected Metrics

Accuracy					**F-Measure**				
	Mean		**P-value**			**Mean**		**P-value**	
	AM	**SM**	**AM**	**SM**		**AM**	**SM**	**AM**	**SM**
AM	0.00	-3.15	1.00	0.00	AM	0.00	-0.02	1.00	0.00
SM	3.15	0.00	0.00	1.00	SM	0.02	0.00	0.00	1.00

- Two different sets of metrics have been considered as the input to develop a model over 45 different object-oriented software.
- There is a significant difference between these approaches.
- The selected metrics yields better results compared to all source code metrics based on mean difference of performance.

Table of Contents

Cost Analysis

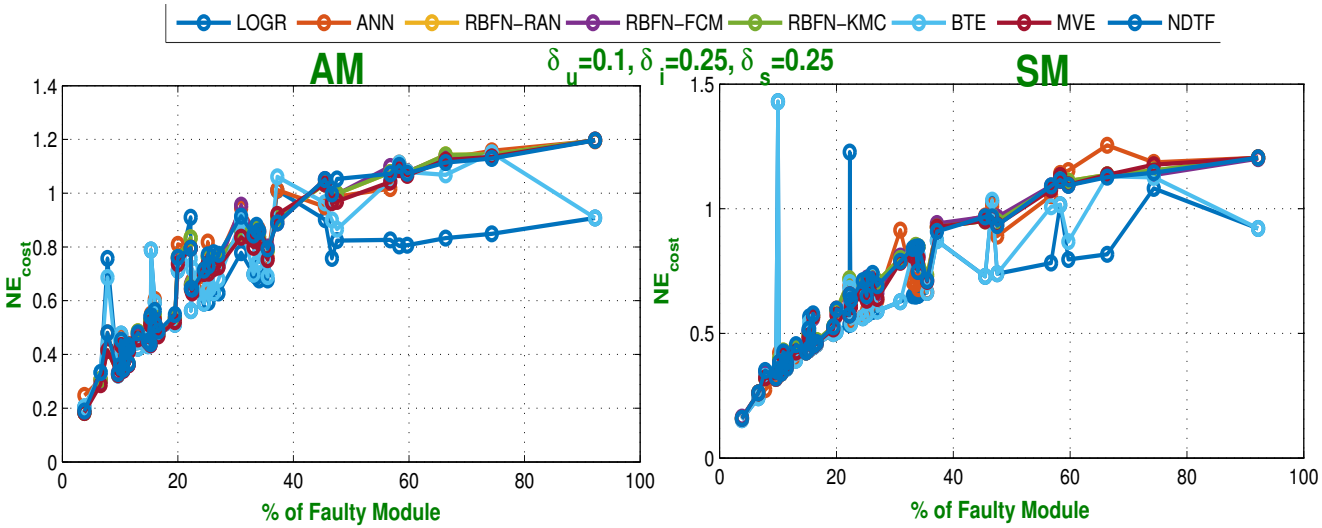


Figure: NE_{cost} for $\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$

As the % faulty classes increases, the fault-prediction technique tends to have a higher value of NE_{cost}, i.e. fault prediction can be useful for the projects with percentage of faulty classes having less than certain threshold.

Table of Contents

Threshold Value

	Const.	Coeff.	Threshold	Const.	Coeff.	Threshold	Const.	Coeff.	Threshold
	$\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$			$\delta_u = 0.25, \delta_i = 0.45, \delta_s = 0.5$			$\delta_u = 0.5, \delta_i = 0.60, \delta_s = 0.65$		
LOGR	-3.62	0.09	38.14	-6.67	0.17	39.64	-7.40	0.27	27.50
ANN	-13.16	0.27	48.75	-11.98	0.33	36.18	-10.88	0.42	25.78
RBFN-RAN	-23.98	0.48	49.75	-11.93	0.30	39.48	-11.56	0.46	25.17
RBFN-FCM	-23.98	0.48	49.75	-16.92	0.42	39.91	-18.80	0.68	27.68
RBFN-KCM	-23.98	0.48	49.75	-11.93	0.30	39.48	-11.56	0.46	25.17
BTE	-4.71	0.08	61.48	-8.47	0.21	39.55	-7.98	0.28	28.00
MVE	-23.98	0.48	49.75	-17.10	0.45	38.40	-12.13	0.44	27.47
NDTF	-24.57	0.51	47.94	-16.92	0.42	39.91	-12.26	0.47	26.23
AM	-7.66	0.14	53.41	-10.68	0.28	37.50	-9.29	0.37	25.23
SM	-7.25	0.13	54.82	-11.23	0.27	41.04	-12.47	0.44	28.10

Threshold Value

- We use logistic regression approach as our dependent variable is dichotomous for the purpose of developing a statistical model to calculate the probability of usefulness of fault prediction techniques (P_{fault}).
- We observe that the fault prediction is useful for the software projects having percentages of faulty classes less than a certain threshold.
- The threshold value expressed in percentage for LOGR and ANN for the case of $\delta_u = 0.1, \delta_i = 0.25, \delta_s = 0.25$ is 38.14 and 48.75 respectively.

Conclusions

Selected set of source code metrics has a high median value in terms of accuracy for all the classifier combinations in comparison to all metrics. This shows that identifying a subset of source code metrics is important.

Ensemble method learning algorithm outperforms individual classifiers.

Fault prediction method is also effective for software projects with a percentage of faulty classes lower than the threshold value (low - 54.82%, medium - 41.04%, high - 28.10%).

References I

[1] Erik Arisholm, Lionel C Briand, and Eivind B Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17, 2010.

[2] Dennis Bahler and Laura Navarro. Methods for combining heterogeneous sets of classifiers. In *17th Natl. Conf. on Artificial Intelligence (AAAI), Workshop on New Research Problems for Machine Learning*, 2000.

[3] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, October 1996.

[4] R Battiti. First and second-order methods for learning between steepest descent and newton's method. *Neural Computation*, 4(2):141–166, 1992.

[5] Shun Bian and Wenjia Wang. On diversity and accuracy of homogeneous and heterogeneous ensembles. *International Journal of Hybrid Intelligent Systems*, 4(2):103–128, 2007.

References II

[6] Jones C. Software quality in 2010: a survey of the state of the art. In *Founder and Chief Scientist Emeritus*, 2010.

[7] Cagatay Catal and Banu Diri. A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346–7354, 2009.

[8] Jie-Cherng Chen and Sun-Jen Huang. An empirical analysis of the impact of software development problem factors on software maintainability. *Journal of Systems and Software*, 82(6):981–992, 2009.

[9] Shyam R. Chidamber and Chris F. Kemerer. Towards a metrics suite for object oriented design. *SIGPLAN Not.*, 26(11):197–211, November 1991.

[10] Shyam R. Chidamber and Chris F. Kemerer. Towards a metrics suite for object oriented design. *SIGPLAN Not.*, 26(11):197–211, November 1991.

[11] Shyamala Doraisamy, Shahram Golzari, Noris Mohd, Md Nasir Sulaiman, and Nur Izura Udzir. A study on feature selection and classification techniques for automatic genre classification of traditional malay music. In *ISMIR*, pages 331–336, 2008.

References III

[12] George Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of machine learning research*, 3:1289–1305, 2003.

[13] Cesare Furlanello, Maria Serafini, Stefano Merler, and Giuseppe Jurman. Entropy-based gene ranking without selection bias for the predictive classification of microarray data. *BMC bioinformatics*, 4(1):1, 2003.

[14] 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.

[15] Ross Huitt and Norman Wilde. Maintenance support for object-oriented programs. *IEEE Transactions on Software Engineering*, 18(12):1038–1044, 1992.

[16] Lov Kumar, Santanu Kumar Rath, and Ashish Sureka. Predicting quality of service (qos) parameters using extreme learning machines with various kernel methods. In *4th International Workshop on Quantitative Approaches to Software Quality*, page 11, 2016.

References IV

[17] Lov Kumar, Santanu Kumar Rath, and Ashish Sureka. Empirical analysis on effectiveness of source code metrics for predicting change-proneness. In *Proceedings of the 10th Innovations in Software Engineering Conference*, pages 4–14. ACM, 2017.

[18] Lov Kumar, Santanu Kumar Rath, and Ashish Sureka. Using source code metrics to predict change-prone web services: A case-study on ebay services. In *Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), IEEE Workshop on*, pages 1–7. IEEE, 2017.

[19] Lov Kumar, Rath Santanu, and Ashish Sureka. An empirical analysis on effective fault prediction model developed using ensemble methods. *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, 2017.

[20] W. Li and S. Henry. Maintenance metrics for the Object-Oriented paradigm. In *Proceedings of First International Software Metrics Symposium*, pages 52–60, 1993.

References V

[21] Wagner S. A literature survey of the quality economics of defect-detection techniques. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering (ISESE)*, pages 194–203, 2006.

[22] Bowen Xu, David Lo, Xin Xia, Ashish Sureka, and Shanping Li. Efspredictor: Predicting configuration bugs with ensemble feature selection. In *Software Engineering Conference (APSEC), 2015 Asia-Pacific*, pages 206–213. IEEE, 2015.

[23] Chi Zhang, Haikun Wei, Liping Xie, Yu Shen, and Kanjian Zhang. Direct interval forecasting of wind speed using radial basis function neural networks in a multi-objective optimization framework. *Neurocomputing*, 2016.