Performance Evaluation Metrics for Software Fault Prediction Studies

Cagatay Catal

Istanbul Kultur University, Department of Computer Engineering, Atakoy Campus, 34156, Istanbul, Turkey, c.catal@iku.edu.tr

Abstract: Experimental studies confirmed that only a small portion of software modules cause faults in software systems. Therefore, the majority of software modules are represented with non-faulty labels and the rest are marked with faulty labels during the modeling phase. These kinds of datasets are called imbalanced, and different performance metrics exist to evaluate the performance of proposed fault prediction techniques. In this study, we investigate 85 fault prediction papers based on their performance evaluation metrics and categorize these metrics into two main groups. Evaluation methods such as cross validation and stratified sampling are not in the scope of this paper, and therefore only evaluation metrics are examined. This study shows that researchers have used different evaluation parameters for software fault prediction until now and more studies on performance evaluation metrics for imbalanced datasets should be conducted.

Keywords: performance evaluation; software fault prediction; machine learning

1 Introduction

Performance evaluation of machine learning-based systems is performed experimentally rather than analytically [33]. In order to evaluate analytically, a formal specification model for the problem and the system itself would be needed. This is quite difficult and inherently non-formalisable for machine learners, which are nonlinear and time-varying [40, 33]. The experimental evaluation of a model based on machine learning is performed according to several performance metrics, such as probability of detection (PD), probability of false alarm (PF), balance, or area under the ROC (Receiver Operating Characteristics) curve. As there are numerous performance metrics that can be used for evaluation, it is extremely difficult to compare current research results with previous works unless the previous experiment was performed by a researcher under the same conditions. Finding a common performance metric can simplify this comparison, but a general consensus is not yet reached. Experimental studies have shown that only a small portion of software modules cause faults in software systems. Therefore, the

majority of software modules are represented with non-faulty labels and the rest are marked with faulty labels during the modeling phase. These kinds of datasets are called imbalanced / unbalanced / skewed, and different performance metrics exist to evaluate the performance of fault prediction techniques that are built on these imbalanced datasets. The majority of these metrics are calculated by using a confusion matrix, which will be explained in later sections. Furthermore, ROC curves are very popular for performance evaluation. The ROC curve plots the probability of a false alarm (PF) on the x-axis and the probability of detection (PD) on the y-axis. The ROC curve was first used in signal detection theory to evaluate how well a receiver distinguishes a signal from noise, and it is still used in medical diagnostic tests [45].

In this study, we investigate 85 software fault prediction papers based on their performance evaluation metrics. In this paper, these metrics are briefly outlined and the current trend is reflected. We included papers in our review if the paper describes research on software fault prediction and software quality prediction. We excluded position papers that do not include experimental results. The inclusion of papers was based on the degree of similarity of the study with our fault prediction research topic. The exclusion did not take into account the publication year of the paper or methods used. We categorized metrics into two main groups: the first group of metrics are used to evaluate the performance of the prediction system, which classifies the module into faulty or non-faulty class; the second group of metrics are used to evaluate the performance of the system, which predicts the number of faults in each module of the next release of a system. Therefore, researchers can choose a metric from one of these groups according to their research objectives. The first group of metrics are calculated by using a confusion matrix. These metrics were identified through our literature review and this set may not be a complete review of all the metrics. However, we hope that this paper will cover the major metrics applied frequently in software fault prediction studies. This paper is organized as follows: Section 2 describes the software fault prediction research area. Section 3 explains the performance metrics. Section 4 presents the conclusions and suggestions.

2 Software Fault Prediction

Software fault prediction is one of the quality assurance activities in Software Quality Engineering such as formal verification, fault tolerance, inspection, and testing. Software metrics [30, 32] and fault data (faulty or non-faulty information) belonging to a previous software version are used to build the prediction model. The fault prediction process usually includes two consecutive steps: training and prediction. In the training phase, a prediction model is built with previous software metrics (class or method-level metrics) and fault data belonging to each

software module. After this phase, this model is used to predict the fault-proneness labels of modules that locate in a new software version. Figure 1 shows this fault prediction process. Recent advances in software fault prediction allow building defect predictors with a mean probability of detection of 71 percent and mean false alarm rates of 25 percent [29]. These rates are at an acceptable level and this quality assurance activity is expected to quickly achieve widespread applicability in the software industry.

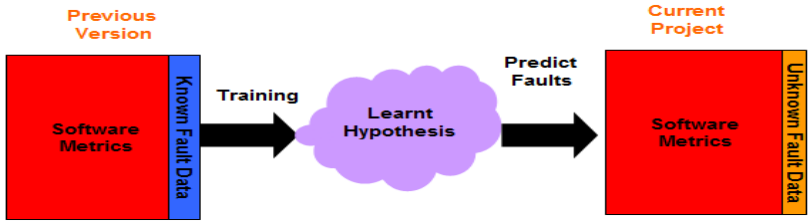

Figure 1
The software fault prediction process [34]

Until now, software engineering researchers have used Case-based Reasoning, Neural Networks, Genetic Programming, Fuzzy Logic, Decision Trees, Naive Bayes, Dempster-Shafer Networks, Artificial Immune Systems, and several statistical methods to build a robust software fault prediction model. Some researchers have applied different software metrics to build a better prediction model, but recent papers [29] have shown that the prediction technique is much more important than the chosen metric set. The use of public datasets for software fault prediction studies is a critical issue. However, our recent systematic review study has shown that only 30% of software fault prediction papers have used public datasets [5].

3 Performance Evaluation Metrics

According to the experimental studies, a majority of software modules do not cause faults in software systems, and faulty modules are up to 20% of all the modules. If we divide modules into two different types, faulty and non-faulty, the majority of modules will belong to the non-faulty class and the rest will be members of the faulty class. Therefore, datasets used in software fault prediction studies are imbalanced. Accuracy parameter cannot be used for the performance evaluation of imbalanced datasets. For example, a trivial algorithm, which marks every module as non-faulty, can have 90% accuracy if the percentage of faulty modules is 10%. Therefore, researchers use different metrics for the validation of software fault prediction models. In this section, the metrics identified during our literature review will be briefly outlined.

3.1 Metrics for Evaluation of Classifiers

Model validation for machine learning algorithms should ensure that data were transformed to the model properly and the model represents the system with an acceptable accuracy. There are several validation techniques for model validation, and the best known one is N-fold cross-validation technique. This technique divides the dataset into N number of parts, and each of them consists of an equal number of samples from the original dataset. For each part, training is performed with (N-1) number of parts and the test is done with that part. Hall and Holmes [17] suggested repeating this test M times to randomize the order each time [29]. Order effect is a critical issue for performance evaluation because certain orderings can improve / degrade performance considerably [13, 29]. In Table 1, a confusion matrix is calculated after N*M cross-validation.

Table 1
Confusion Matrix

	NO (Prediction)	YES (Prediction)
NO (Actual)	True Negative (TN) A	False Positive (FP) B
YES (Actual)	False Negative (FN) C	True Positive (TP) D

Columns represent the prediction results and rows show the actual class labels. Faulty modules are represented with the label YES, and non-faulty modules are represented with the label NO. Therefore, diagonal elements (TN, TP) in Table 1 show the true predictions and the other elements (FN, FP) reflect the false predictions. For example, if a module is predicted as faulty (YES) even though it is a non-faulty (NO) module, this test result is added to the B cell in the table. Therefore, number B is incremented by 1. After M*N tests, A, B, C, and D values are calculated. In the next subsections, these values (A, B, C, D) will be used to compute the performance evaluation metrics.

3.1.1 PD, PF, Balance

The equations used to calculate probability of detection (PD), probability of false alarm (PF), and accuracy metrics are shown in Formulas 1, 2, and 3 respectively. The other term used for PD metric is recall.

$$\text{PD} = \text{recall} = \frac{D}{C + D} = \frac{TP}{TP + FN} \quad (1)$$

$$\text{PF} = \frac{B}{A + B} = \frac{FP}{FP + TN} \quad (2)$$

$$\text{Accuracy} = \frac{A + D}{(A + B + C + D)} \tag{3}$$

Balance metrics is the Euclidean distance between (0, 1) and (PF, PD) points. PD, accuracy, and balance parameters should be maximized and PF metrics should be minimized for fault predictors. Menzies et al. [29] reported that the best fault predictors provide 71% of PD and 25% of PF values. They used PD, PF, and balance parameters as the performance evaluation metrics in this study. Turhan and Bener [38] showed that the independence assumption in the Naive Bayes algorithm is not detrimental with principal component analysis (PCA) pre-processing, and they used PD, PF, and balance parameters in their study.

3.1.2 G-mean1, G-mean2, F-measure

Some researchers use G-mean1, G-mean2, and F-measure metrics for the evaluation of prediction systems, which are built on imbalanced datasets. Formulas 6, 7, and 8 show how to calculate these measures, respectively. Formula 4 is used for precision parameter and True Negative Rate (TNR) is calculated by using Formula 5. The formula for recall is given in Formula 1.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4}$$

$$\text{True Negative Rate (TNR)} = \frac{TN}{TN + FP} \tag{5}$$

$$\text{G-mean1} = \sqrt{\text{Precision} * recall} \tag{6}$$

$$\text{G-mean2} = \sqrt{recall * TNR} \tag{7}$$

$$\text{F-measure} = \frac{2\ (\text{recall} * \text{Precision})}{\text{recall} + \text{Precision}} \tag{8}$$

Ma et al. [26] used G-mean1, G-mean2, and G-mean3 to benchmark several machine learning algorithms for software fault prediction. They sorted algorithms according to their performance results for each metric and marked the top three algorithms for each metric. They identified the algorithm that provides G-mean1, G-mean2, and F-measure values in the top three. According to this study, Balanced Random Forests is the best algorithm for software fault prediction problems. Furthermore, they reported that boosting, rule set, and single tree classifiers do not provide acceptable results even though these algorithms have been used in literature. Koru and Liu [23] evaluated the performance of classifiers according to the F-measure value. Arisholm et al. [1] built 112 fault prediction models and compared them according to precision, recall, accuracy, Type-I error, Type-II error, and AUC parameters. The following sections will explain AUC, Type-I, and Type-II errors.

3.1.3 AUC

Receiver Operating Characteristics (ROC) curves can be used to evaluate the performance of software fault prediction models. In signal detection theory, a ROC curve is a plot of the sensitivity vs. (1-specificity) and it can also be represented by plotting the probability of false alarm on the X-axis and the probability of detection on the Y-axis. This curve must pass through the points (0, 0) and (1, 1) [29]. The important regions of ROC curve are depicted in Figure 2. The ideal position on ROC curve is (0, 1) and no prediction error exists at this point. A line from (0, 0) to (1, 1) provides no information and therefore the area under ROC curve value (AUC) must be higher than 0.5. If a negative curve occurs, this means that the performance of this classifier is not acceptable. A preferred curve is shown in Figure 2. The cost-adverse region has low false alarm rates and is suitable if the validation & verification budget is limited. In the risk-adverse region, even though the probability of detection is high, the probability of false alarm is also high, and, therefore, cost is higher. For mission critical systems, a risk-adverse region is chosen and for business applications, a cost-adverse region is more suitable.

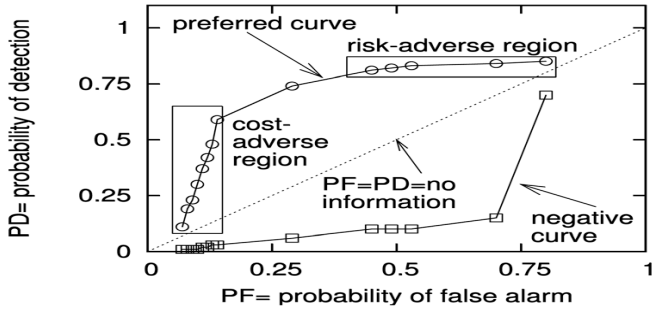

Figure 2
Regions of ROC curve [29]

The area under the ROC curve (AUC) is a widely used performance metric for imbalanced datasets. Ling et al. [25] proposed the usage of an AUC parameter to evaluate the classifiers and showed that AUC is much more appropriate than accuracy for balanced and imbalanced datasets. Van Hulse et al. [39] applied an AUC metric to evaluate the performance of 11 learning algorithms on 35 datasets. In addition to this metric, they also utilized Kolmogorov-Smirnov (K/S) statistics [18], geometric mean, F-measure, accuracy, and true positive rate (TPR) parameters. They stated that AUC and K/S parameters measure the capability of the classifier and showed AUC values of algorithms in tables. Li et al. [24], and Chawla and Karakoulas [7] used an AUC parameter for unbalanced datasets. For a competition in "11th Pacific-Asia Conference on Knowledge Discovery and Data Mining" (PAKDD2007), performance evaluations for an imbalanced dataset were

performed according to AUC values, and the model that provides 70.01% of AUC value was selected as the best algorithm. Catal and Diri [6] examined nine classifiers and compared their performance according to the AUC value. Mende and Koschke [28] used an AUC parameter to compare classifiers on thirteen datasets.

3.1.4 Sensitivity, Specificity, J Coefficient

El-Emam et al. [11] proposed the usage of the J parameter to measure the accuracy of binary classifiers in software engineering. The J coefficient was first used in medical research [41]; it is calculated by using sensitivity and specificity parameters. El-Emam et al. [12] used the J coefficient for performance evaluation of algorithms. Sensitivity, specificity, and the J parameter are calculated by using Formulas 9, 10, and 11 respectively.

$$\text{Sensitivity} = \frac{D}{C + D} = \frac{TP}{TP + FN} \tag{9}$$

$$\text{Specificity} = \frac{A}{A + B} = \frac{TN}{TN + FP} \tag{10}$$

$$J = \text{sensitivity} + \text{specificity} - 1 \tag{11}$$

Sensitivity measures the ratio of actual faulty modules which are correctly identified and specificity measures the ratio of non-faulty modules which are correctly identified.

3.1.5 Type-I error, Type-II error, Overall Misclassification Rate

Some researchers used Type-I error and Type-II error parameters to evaluate the performance of fault prediction models [42, 15, 35, 1, 2]. The overall misclassification rate parameter takes care of these two error parameters. Formulas 12, 13, and 14 are used to calculate the Type-I error, Type-II error, and overall misclassification rate respectively. If a non-faulty module is predicted as a faulty module, a Type-I error occurs, and if a faulty module is predicted as a non-faulty module, a Type-II error occurs. A Type-II error is more significant than a Type-I error because faulty modules cannot be detected in that case.

$$\text{Type-I error} = \frac{B}{A + B + C + D} = \frac{FP}{TN + FP + FN + TP} \tag{12}$$

$$\text{Type-II error} = \frac{C}{A + B + C + D} = \frac{FN}{TN + FP + FN + TP} \tag{13}$$

$$\text{Overall misclassification rate} = \frac{C + B}{A + B + C + D} \tag{14}$$

3.1.6 Correctness, Completeness

Correctness and completeness parameters were used for the evaluation of fault prediction models [4, 44, 9, 16, 27]. Formulas 15 and 16 show how to calculate correctness and completeness measures.

$$\text{Correctness} = \frac{D}{B + D} = \frac{TP}{FP + TP} \tag{15}$$

$$\text{Completeness} = \frac{D}{C + D} = \frac{TP}{FN + TP} \tag{16}$$

3.1.7 FPR, FNR, Error

The false positive rate (FPR), the false negative rate (FNR), and error parameters are used for performance evaluation [41, 43].

$$FPR = \frac{FP}{FP + TN} \tag{17}$$

$$FNR = \frac{FN}{FN + TP} \tag{18}$$

$$Error = \frac{FN + FP}{TP + FP + FN + TN} \tag{19}$$

These three performance indicators should be minimized, but there is a trade-off between the FPR and FNR values. The FNR value is much more crucial than the FPR value because it quantifies the detection capability of the model on fault-prone modules and high FNR values indicate that a large amount of fault-prone modules cannot be captured by the model before the testing phase. Therefore, users will probably encounter these problems in the field and the nondetected faulty modules can cause serious faults or even failures. On the other hand, a model having high FPR value will simply increase the testing duration and test efforts.

3.1.8 Cost Curve

Jiang et al. [19] recommended adopting cost curves for the fault prediction performance evaluation. This is the first study to propose cost curves for performance evaluation of fault predictors and it is not yet widely used. However, it is not easy to determine the misclassification cost ratio and the selection of this parameter can make the model debatable. Drummond and Holte [10] proposed cost curves to visualize classifier performance and the cost of misclassification was included in this technique. Cost curve plots the probability cost function on the x-axis and the normalized expected misclassification cost on the y-axis. Details of this approach will not be given here due to length considerations, and

readers may apply to papers by Jiang et al. [19] or Drummond and Holte [10] to learn the details of this approach. However, this approach is not widely used in the software fault prediction research area.

3.2 Metrics for the Evaluation of Predictors

Some researchers predict the number of faults in each module of the next release of a system, and the modules' classification is performed according to the number of faults. Modules are sorted in descending order with respect to the number of faults, and the modules which should be tested rigorously are identified according to the available test resources.

3.2.1 Average Absolute Error, Average Relative Error

Average absolute error and average relative error parameters have been used as performance evaluation metrics by numerous researchers for software quality prediction studies [22, 20, 21, 14]. Formulas 20 and 21 show how to calculate average absolute error (AAE) and average relative error (ARE) parameters, respectively. The actual number of faults is represented by y_i, while y_j represents the predicted number of faults, and n shows the number of modules in the dataset.

$$\text{AAE} = \frac{1}{n}\sum_{i=1}^{n} \left| y_i - y_j \right| \tag{20}$$

$$\text{ARE} = \frac{1}{n}\sum_{i=1}^{n} \left| \frac{y_i - y_j}{y_i + 1} \right| \tag{21}$$

3.2.2 R^2

R^2 measures the power of correlation between predicted and actual number of faults [37]. Another term for this parameter is the *coefficient of multiple determination*, and this parameter is widely used in studies that predict the number of faults. Many researchers have applied this parameter in their studies [9, 36, 8, 37, 31, 3]. This metric's value should be near to 1 if the model is to be acceptable, and Formula 22 is used to calculate this parameter. The actual number of faults is represented by y_i, $\hat{y}_i$ represents the predicted number of faults, and $\overline{y}$ shows the average of fault numbers.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \overline{y})^2} \tag{22}$$

Conclusions

The use of different evaluation parameters prevents the software engineering community from easily comparing research results with previous works. In this study, we investigated 85 fault prediction papers based on their performance evaluation metrics and categorized these metrics into two main groups. The first group of metrics are used for prediction systems that classify modules into a faulty or non-faulty module and the second group of metrics are applied to systems that predict the number of faults in each module of the next release of a system. This study showed that researchers have used numerous evaluation parameters for software fault prediction up to now, and the selection of common evaluation parameters is still a critical issue in the context of software engineering experiments. From the first group, the most common metric for software fault prediction research is the area under ROC curve (AUC). The AUC value is only one metric and it is not a part of the metric set. Therefore, it is easy to compare several machine learning algorithms by using this parameter. In addition to AUC, PD, and PF, balance metrics are also widely used. In this study, we suggest using the AUC value to evaluate the performance of fault prediction models. From the second group of metrics, R^2 and AAE / ARE can be used to ensure the performance of the system that predicts the number for faults. We suggest the following changes in software fault prediction research:

- *Conduct more studies on performance evaluation metrics for software fault prediction.* Researchers are still working on finding a new performance evaluation metric for fault prediction [19], but we need more research in this area because this software engineering problem is inherently different than the other imbalanced dataset problems. For example, it is not easy to determine the misclassification cost ratio (Jiang et al., 2008) and therefore, using cost curves for evaluation is still not an easy task.
- *Apply a widely used performance evaluation metric.* Researchers would like to be able to easily compare their current results with previous works. If the performance metric of previous studies is totally different than the widely used metrics, that makes the comparison difficult.

References

[1] E. Arisholm, L. C. Briand, and E. B. Johannessen, A Systematic and Comprehensive Investigation of Methods to Build and Evaluate Fault Prediction Models, Journal of Systems and Software 83 (1) (2010) 2-17

[2] Y. Bingbing, Y. Qian, X. Shengyong, and G. Ping, Software Quality Prediction Using Affinity Propagation Algorithm, Proc. IJCNN 2008, 2008, pp. 1891-1896

[3] D. Binkley, H. Feild, D. Lawrie, M. Pighin, Software Fault Prediction Using Language Processing, Proc. Testing: Academic and industrial Conference Practice and Research Techniques - MUTATION, TAICPART-MUTATION 2007, Washington, DC, 2007, pp. 99-110

[4] L. C. Briand, V. Basili, C. Hetmanski, Developing Interpretable Models with Optimized Set Reduction for Identifying High Risk Software Components, IEEE Transactions on Software Engineering 19 (11) (1993) 1028-1044

[5] C. Catal, B. Diri, A Systematic Review of Software Fault Prediction Studies, Expert Systems with Applications 36 (4) (2009a) 7346-7354

[6] C. Catal, B. Diri, Investigating the Effect of Dataset Size, Metrics Sets, and Feature Selection Techniques on Software Fault Prediction Problem, Inf. Sci. 179 (8) (2009b) 1040-1058

[7] N. V. Chawla, G. J. Karakoulas, Learning from Labeled and Unlabeled Data. An Empirical Study across Techniques and Domains, Journal of Artificial Intelligence Research, 23 (2005) 331-366

[8] G. Denaro, Estimating Software Fault-Proneness for Tuning Testing Activities, Proc. 22nd Int'l Conf. on Soft. Eng., Limerick, Ireland, 2000, pp. 704-706

[9] G. Denaro, M. Pezzè, S. Morasca, Towards Industrially Relevant Fault-Proneness Models, International Journal of Software Engineering and Knowledge Engineering 13 (4) (2003) 395-417

[10] C. Drummond, R. C. Holte, Cost Curves: An Improved Method for Visualizing Classifier Performance, Machine Learning 65 (1) (2006) 95-130

[11] K. El-Emam, S. Benlarbi, N. Goel, Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components, Technical Report, National Research Council of Canada, NRC/ERB-1058, Canada, 1999

[12] K. El-Emam, W. Melo, J. C. Machado, The Prediction of Faulty Classes Using Object-oriented Design Metrics, Journal of Systems and Software 56 (1) (2001) 63-75

[13] D. Fisher, L. Xu, N. Zard, Ordering Effects in Clustering, Proc. Ninth Int'l Conf. Machine Learning, 1992

[14] K. Gao, T. M. Khoshgoftaar, A Comprehensive Empirical Study of Count Models for Software Fault Prediction, IEEE Transactions for Reliability 56 (2) (2007) 223-236

[15] P. Guo, M. R. Lyu, Software Quality Prediction Using Mixture Models with EM Algorithm, Proc. 1st Asia-Pacific Conference on Quality Software, Hong Kong, 2000, pp. 69-80

[16] T. Gyimothy, R. Ferenc, I. Siket, Empirical Validation of Object-oriented Metrics on Open Source Software for Fault Prediction, IEEE Transactions on Software Engineering 31 (10) (2005) 897-910

[17] M. Hall, G. Holmes, Benchmarking Attribute Selection Techniques for Discrete Class Data Mining, IEEE Trans. Knowledge and Data Eng. 15 (6) (2003) 1437-1447

[18] D. J. Hand, Good Practice in Retail Credit Scorecard Assessment, Journal of the Operational Research Society 56 (2005) 1109-1117

[19] Y. Jiang, B. Cukic, T. Menzies, Cost Curve Evaluation of Fault Prediction Models, Proc. 19th International Symposium on Software Reliability Engineering, IEEE Computer Society, Washington, DC, 2008, pp. 197-206

[20] T. M. Khoshgoftaar, N. Seliya, Tree-based Software Quality Estimation Models for Fault Prediction. 8th IEEE Symposium on Software Metrics. Ottawa, Canada, 2002, pp. 203-215

[21] T. M. Khoshgoftaar, E. Geleyn, K. Gao, An Empirical Study of the Impact of Count Models Predictions on Module-Order Models, Proc. 8th Int'l Symp. on Software Metrics, Ottawa, Canada, 2002, pp. 161-172

[22] T. M. Khoshgoftaar, N. Seliya, N. Sundaresh, An Empirical Study of Predicting Software Faults with Case-based Reasoning, Software Quality Journal 14 (2) (2006) 85-111

[23] A. G. Koru, H. Liu, An Investigation of the Effect of Module Size on Defect Prediction Using Static Measures, Proc. Workshop on Predictor Models in Software Engineering, St. Louis, Missouri, 2005, pp. 1-5

[24] X. Li, L. Wang, E. Sung, AdaBoost with SVM-based Component Classifiers, Eng. Appl. Artif. Intelligence 21 (5) (2008) 785-795

[25] C. X. Ling, J. Huang, H. Zhang, AUC: A Better Measure than Accuracy in Comparing Learning Algorithms, Canadian Conference on Artificial Intelligence, Halifax, Canada, 2003, pp. 329-341

[26] Y. Ma, L. Guo, B. Cukic, A Statistical Framework for the Prediction of Fault-Proneness, Advances in Machine Learning Application in Software Engineering, Idea Group Inc., 2006, pp. 237-265

[27] A. Mahaweerawat, P. Sophasathit, C. Lursinsap. Software Fault Prediction Using Fuzzy Clustering and Radial Basis Function Network, Proc. International Conference on Intelligent Technologies, Vietnam, 2002, pp. 304-313

[28] T. Mende, R. Koschke, Revisiting the Evaluation of Defect Prediction Models, Proc. 5th international Conference on Predictor Models in Software Engineering, Vancouver, Canada, 2009, pp. 1-10

[29] T. Menzies, J. Greenwald, A. Frank, Data Mining Static Code Attributes to Learn Defect Predictors, IEEE Transactions on Software Engineering 32 (1) (2007) 2-13

[30] S. Misra, Evaluation Criteria for Object-oriented Metrics, Acta Polytechnica Hungarica 8(5) (2011) 109-136

[31] A. P. Nikora, J. C. Munson, Building High-Quality Software Fault Predictors, Software Practice and Experience 36 (9) (2006) 949-969

[32] O. T. Pusatli, S. Misra, Software Measurement Activities in Small and Medium Enterprises: An Empirical Assessment, Acta Polytechnica Hungarica 8(5) (2011) 21-42

[33] F. Sebastiani, Machine Learning in Automated Text Categorization, ACM Comput. Surv. 34 (1) (2002) 1-47

[34] N. Seliya, Software Quality Analysis with Limited Prior Knowledge of Faults. Graduate Seminar, Wayne State University, Department of Computer Science, 2006

[35] N. Seliya, T. M. Khoshgoftaar, Software Quality Estimation with Limited Fault Data: A Semi-supervised Learning Perspective, Software Quality Journal 15 (3) (2007) 327-344

[36] M. M. Thwin, T. Quah, Application of Neural Networks for Software Quality Prediction Using Object-oriented Metrics, Proc. 19th International Conference on Software Maintenance, Amsterdam, The Netherlands, 2003, pp. 113-122

[37] P. Tomaszewski, L. Lundberg, H. Grahn, The Accuracy of Early Fault Prediction in Modified Code, Proc. 5th Conference on Software Engineering Research and Practice in Sweden, Västerås, Sweden, 2005, pp. 57-63

[38] B. Turhan, A. Bener, Analysis of Naive Bayes' Assumptions on Software Fault Data: An Empirical Study, Data Knowl. Eng. 68 (2) (2009) 278-290

[39] J. Van Hulse, T. M. Khoshgoftaar, A. Napolitano, Experimental Perspectives on Learning from Imbalanced Data, 24th International Conference on Machine Learning, Corvalis, Oregon, 2007, pp. 935-942

[40] H. Wang, Y. Chen, and Y. Dai, A Soft Real-Time Web News Classification System with Double Control Loops, Proc. WAIM 2005, 2005, pp. 81-90

[41] W. Youden, Index for Rating Diagnostic Tests, Cancer 3 (1) (1950) 32-35

[42] X. Yuan, T. M. Khoshgoftaar, E. B. Allen, K. Ganesan, An Application of Fuzzy Clustering to Software Quality Prediction, Proc. 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, Richardson, Texas, 2000, p. 85

[43] S. Zhong, T. M. Khoshgoftaar, N. Seliya. Unsupervised Learning for Expert-based Software Quality Estimation, Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004, pp. 149-155

[44] Y. Zhou, H. Leung, Empirical Analysis of Object-oriented Design Metrics for Predicting High and Low Severity Faults, IEEE Transactions on Software Engineering 32 (10) (2006) 771-789

[45] M. J. Zolghadri, E. G. Mansoori, Weighting Fuzzy Classification Rules Using Receiver Operating Characteristics (ROC) Analysis, Inf. Sci. 177 (11) (2007) 2296-2307