Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 3714 – 3723
© TÜBİTAK
doi:10.3906/elk-1409-137

The use of cross-company fault data for the software fault prediction problem

Çağatay ÇATAL[*]
Department of Computer Engineering, İstanbul Kültür University, Ataköy Campus, İstanbul, Turkey

Received: 20.09.2014 • **Accepted/Published Online:** 05.06.2015 • **Final Version:** 20.06.2016

Abstract: We investigated how to use cross-company (CC) data in software fault prediction and in predicting the fault labels of software modules when there are not enough fault data. This paper involves case studies of NASA projects that can be accessed from the PROMISE repository. Case studies show that CC data help build high-performance fault predictors in the absence of fault labels and remarkable results are achieved. We suggest that companies use CC data if they do not have any historical fault data when they decide to build their fault prediction models.

Key words: Metrics values, defect prediction, cross-company data

1. Introduction

Software fault prediction is a quality-assurance technique within the software quality engineering discipline and includes tasks such as fault tolerance, testing, and inspection [1]. However, software testing is not familiar to most software practitioners. We need more practical tools for the widespread usage of this technique.

According to experimental studies, only a small number of modules cause software faults [2]. Therefore, the identification of these modules helps quality assurance groups or project managers to allocate testing resources in an efficient manner. One approach might be to test fault-prone modules in detail; these modules can be identified by using a fault prediction model. This policy will probably detect more faults in a short period of time. Since the testing phase is not unlimited, but rather must be finished within an allocated time and budget, fault prediction models help companies to identify problematic code blocks that negatively affect the maintainability and reusability characteristics of the product.

Artificial neural networks, tree-based methods, fuzzy logic, genetic programming, analogy-based approaches, statistical procedures, and several biology-inspired computing paradigms, such as artificial immune systems, have been used to build fault prediction models so far [3]. Early studies did not use public datasets and it was very difficult to assess the relative performance of each model in literature. Later, researchers recognized the use of public datasets and comparative studies primarily used NASA public datasets from the PROMISE repository. This helped the software engineering community to identify high-performance machine learners.

Researchers mostly built fault prediction models that used metrics and fault data. These kinds of models, which implement different types of learning algorithms, are known in the field of machine learning as supervised learning approaches. When there are no prior fault data, we cannot use supervised learning methods.

It is known that fault prediction is more useful for high-risk projects in which fault data are generally recorded. Therefore, practitioners might assume that all high-risk projects will always have fault data for every

[*]Correspondence: c.catal@iku.edu.tr

case. However, this assumption is not true in the case of the first release of a software product. In this case, there are no fault data and it is not possible to use within-company data in this context. The question in this scenario is whether we can use cross-company (CC) data to predict the fault-proneness labels of our modules or not. Also, when we consider the global nature of software engineering, and the fact that software is developed in different geographical regions, some companies that produce outsourced software might not collect defect data for several reasons. For example, a subcontractor company might believe that recording every fault in the bug-tracking system creates the perception of a poorly developed software system. Therefore, they might not update the bug-tracking system correctly and very few labeled instances might exist for the project. As we see in this scenario, if the company does not collect fault data for a version, there will not be any historical data for modeling.

For the project cost estimation problem [4] in software engineering, some researchers applied datasets from other companies. Although software fault prediction and cost estimation problems are quite different from many perspectives, some researchers published satisfactory results prior to our research.

Also, as Turhan et al. stated [5], people do not want to collect new data, even if these projects belong to NASA; instead, they just want to use the old data. Although these projects were developed for NASA, they were developed by different NASA contractors that are in different geographical regions in the United States. Turhan et al. [5] used 7 datasets from NASA and stated that they treated these projects as different projects from 7 different companies. They also emphasized that it is possible to examine the CC defect prediction, even if experiments were performed on NASA datasets. These are very different projects, such as satellite instrumentation, flight control modules, ground control systems, video guidance systems, and zero-gravity experiments.

Turhan et al. [5] published the first paper on CC dataset usage for the fault prediction problem and reported that it is possible to use CC data if the nearest-neighborhood filtering approach is used. Datasets were taken from NASA contractors [5] after extensive work over 5 years and these datasets were used as CC data in their study.

Later, Zimmermann et al. [6] worked on 622 cross-project fault predictions for several software applications. In their study, four projects were from the open source community and the rest were from the commercial applications of Microsoft Corporation. Contrary to Turhan et al. [5], Zimmermann et al. [6] stated that projects from a similar domain might not be enough to build high-performance models. They concluded that domain, code data, and process should also be investigated for a true fault prediction model.

This study is another investigation of CC data to design a fault prediction model, and the results of our approach are very promising. We preferred public NASA datasets because they are publicly available in the PROMISE repository, they were carefully collected over 5 years from several NASA contractors, and they are not proprietary applications. In this study, we designed and implemented a fault prediction technique that uses metrics threshold values instead of machine learner-based predictive models.

The motivation behind this threshold-based approach is the practical considerations of metrics threshold values, as explained by Shatnawi et al. [7]. Shatnawi et al. [7] stated that building and running a metrics model on a daily basis is impractical for software engineers, and the use of threshold values is relatively easier than building metrics-based predictive models. In their study, they proposed a new metric threshold calculation approach that uses receiver operating characteristic (ROC) curves. The research questions are as follows:

- Can we use CC data to calculate metrics threshold values and apply these threshold values in a comprehensive software fault prediction technique instead of complex machine learning-based approaches?

- Is the proposed threshold-based approach better than the popular naive Bayes-based models?

The proposed fault prediction technique is uncomplicated, but it is comprehensive and high-performance. In the proposed technique, when the majority of the metric values exceed their corresponding threshold values, that module is classified as fault-prone.

The contributions of this paper are two-fold:

1. A new comprehensive and high-performance fault prediction technique based on metrics threshold values.

   The performance of our model is better than the performance of naive Bayes with logarithmic filter in terms of the probability of detection (PD) value. Area under ROC curve (AUC) values are similar. The PD is slightly better than the naive Bayes-based approach's PD and the probability of false alarm (PF) is slightly worse. For mission-critical systems that require high PD values, the proposed approach is much more suitable.

2. New evidence that CC data are useful for building fault prediction models.

   The method first calculates metrics threshold values based on ROC curves and uses these values to classify modules into two groups. Even though a different approach than the naive Bayes-based prediction technique has been applied, CC data were very useful in this context.

The remainder of the paper is structured as follows. Section 2 introduces software fault prediction and related practical problems. Section 3 presents related work. Section 4 describes the methodology and the threshold-based technique. Section 5 presents empirical case studies. Section 6 contains the conclusion and suggestions for future work.

2. Absence of fault labels

Software fault prediction models require software metrics that are collected from version control systems, such as Subversion, and fault data belonging to the previous version of software from change management or bug-tracking systems, such as Bugzilla.

Software fault prediction without prior fault data is shown in Figure 1. To solve this challenging problem, Zhong et al. [8] developed a technique based on clustering algorithms. Modules were separated into clusters with clustering algorithms. After that, an experienced engineer classified each cluster into two groups, fault-prone and nonfault-prone, by examining the statistical features of each cluster. This method combined clustering and expert-based software quality estimation in order to solve the problem of missing fault labels. However, finding an expert to label the modules is difficult for many companies. During the labeling process, statistical features of the clusters have been investigated. According to our literature survey, Zhong et al. [8] worked on this problem for the first time, and still we see a huge potential for further research. In this study, we aimed to remove the expert dependency and design a totally automated approach [9].

The design and use of complex machine-learning algorithms for practical software fault prediction problems is very frustrating for software engineers and we aimed to build comprehensive prediction models for cases where there are no, or limited, fault data. Existing limited fault data can be discarded and the problem can easily be transformed into a no-fault-data problem. We removed the class labels of the NASA datasets to simulate the absence of fault data before the prediction is performed.

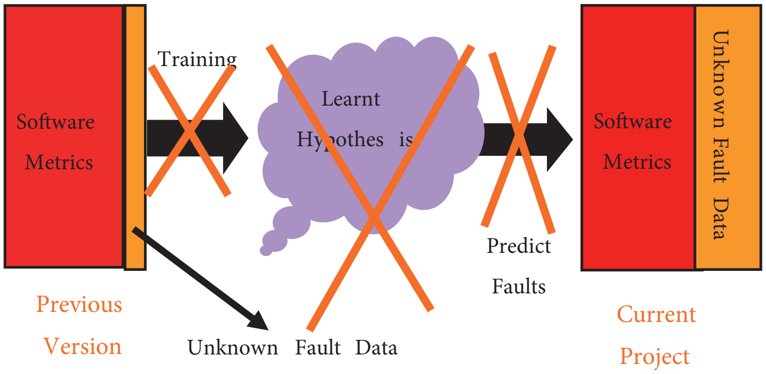

Figure 1. Fault prediction without fault data.

3. Related work

So far, most of the models reported in the literature used within-company data as previous fault data [10]. Although most companies do not collect within-company data in practice, there are only a few studies that explain what to do when there are not enough fault data. One approach is to use CC data as previous fault data and run classification algorithms for prediction.

Turhan et al. [5] worked on seven datasets that were collected from NASA's contractors. When one dataset is tested, the remaining datasets are used as CC data. A popular classification algorithm known as naive Bayes has been applied to the problem. Zimmerman et al. [6] used open source projects and commercial software from Microsoft Corporation for defect prediction. Most of the authors in Zimmerman et al.'s [6] paper work in Microsoft Research; therefore, they could easily access the data of applications developed by Microsoft Corporation.

Another approach is to use unsupervised learning algorithms in the absence of fault information. Zhong et al. [8] applied clustering methods to solve the problem, as explained in the previous section. Later, Zhong et al. [11] applied it to a large dataset known as JM1. Seliya et al. [12] proposed a semisupervised clustering model that applied a K-means clustering approach. This approach helped the expert much more than the unsupervised learning algorithm. However, it is difficult to identify the number of clusters and it is still dependent on the expert.

The expectation-maximization method was applied by Seliya et al. [13] to limited fault data problems, demonstrating that this prediction model is better than the classification algorithms. However, semisupervised classification algorithms do not always perform better [14]. In fact, the use of unlabeled data in the scope of semisupervised classification may degrade the classifier performance if data do not conform to the model assumptions, as explained in previous section.

Shatnawi et al. [7] proposed a new approach that uses ROC curves to calculate metric threshold values. The threshold calculation approach in our study differs from that of Shatnawi et al. [15]. Researchers in the machine learning community apply the AUC metric for benchmarking [16]. In their study, they selected the candidate threshold having maximum values for specificity and sensitivity as the real threshold value.

In our study, we computed an AUC that passed from the following points: (PD, PF), (0, 0), and (1, 1). We selected the candidate threshold that had the highest AUC value for the real threshold value. We used Table 1 to calculate the confusion matrix of candidate thresholds. When the module's value was smaller than the candidate's value, this module was predicted as nonfault-prone [15]. Otherwise, this module was classified as fault-prone, according to Table 1. We already had information about the actual labels of modules, and we easily computed (PD, PF) pairs for candidate thresholds.

Table 1. Error matrix for candidate metric [15].

Predicted	Actual		
		Fault-prone	Not fault-prone
	Metric ≥ threshold	True positive	False positive
	Metric <threshold	False negative	True negative

Eqs. (1) and (2) show how to calculate PD and PF. "Each pair, (PD, PF), represents the classification performance of the threshold value that produces this pair. The pair that produces the best classification performance is the best threshold value to use in practice" [7].

$$\mathrm{PD} = \frac{TP}{TP + FN} \tag{1}$$

$$\mathrm{PF} = \frac{FP}{FP + TN} \tag{2}$$

Shatnawi et al. [7] decided to select a candidate threshold that had a maximum value of PD and a minimum value of PF for the best threshold value [15]. We observed that this might not always be possible. Also, it is possible to see that one candidate may reach the highest PD value, and another candidate may provide the lowest PF [15]. Therefore, we decided to change our threshold calculation policy.

According to our threshold calculation approach, the candidate with the highest AUC value was selected as the best threshold value. In addition, the AUC for the selected candidate value could not be less than 0.5. This approach was used to calculate the threshold value of each metric.

After calculating the threshold values of metrics in datasets, we used these values in the threshold-based fault prediction approach. The training dataset is the collection of six datasets, with the test dataset excluded. For example, if the prediction was done for the KC2 dataset, the training dataset of that experiment was the collection of datasets KC3, CM1, PC1, MW1, and MC2.

The most popular classification algorithm in machine learning is naive Bayes, although it is not a very complicated method [17]. It has been previously shown that naive Bayes is very good at predicting faults when it is applied with a logarithmic filter for datasets collected from NASA projects [18].

Researchers first took the logarithm of metrics in datasets and applied the naive Bayes method on this converted dataset. Menzies et al. [19] showed that only 50 modules might be enough to build an acceptable prediction model, and that naive Bayes provided the best performance, as in their previous study [18].

They stated that we must not focus on new machine learning methods. Instead, we must take into account the information content of the available datasets. In their study, it was concluded that using a small dataset instead of a large one does not affect the performance dramatically.

4. Methodology

This section introduces the proposed threshold-based fault prediction approach. Our threshold-based fault prediction approach is straightforward. After metrics threshold values are calculated, a module can be predicted as fault-prone or not fault-prone depending on the module's metrics values. In Figure 2, the flowchart of our approach is depicted.

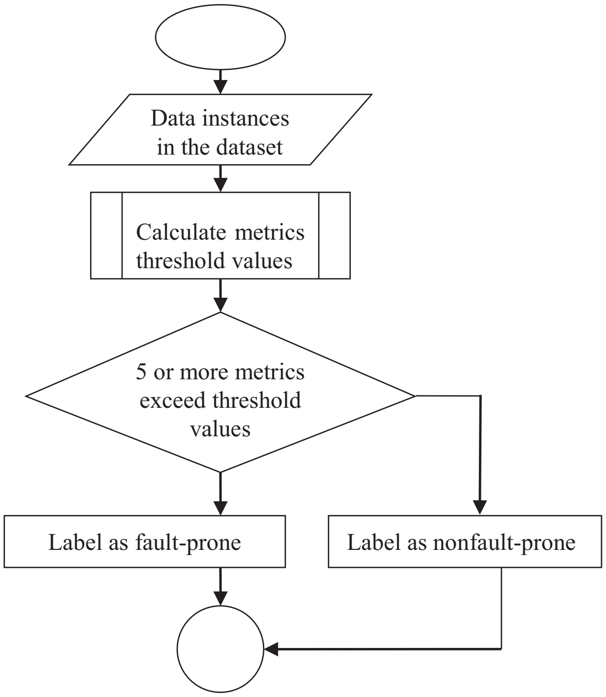
Data instances in the dataset
Calculate metrics threshold values
5 or more metrics exceed threshold values
Label as fault-prone
Label as nonfault-prone

Figure 2. Our fault prediction approach.

When the majority of the software metrics exceed their threshold values, that module is classified as fault-prone. For nine metrics, the majority value is five. Therefore, when at least five software metrics values exceed their corresponding threshold values, that module is labeled as fault-prone. Otherwise, it is labeled as not fault-prone. NASA datasets include 21 metrics, but most researchers prefer 13 metrics because the remaining ones are derived metrics, which are not useful for prediction [15]. We eliminated four software metrics (count of lines of comments, count of blank lines, lines of comment and code, Halstead line count) because the threshold values were not meaningful for the first three of them and the last metric was very similar to McCabe lines of code. Therefore, we worked with nine metrics for the experiments.

These metrics are cyclomatic complexity, lines of code, unique operands, essential complexity, design complexity, unique operators, total operands, total operators, and branch count. The experimental studies showed that this threshold-based fault prediction approach is effective in detecting fault-prone modules. The number of software metrics can change from one development environment to another and the main motivation behind this approach is the use of software metrics threshold values. Even though the technique seems limited to these nine metrics, this approach can be used with different metrics subsets.

5. Experimental results

In this section, experimental results are explained. Section 5.1 will show the datasets used in this study and Section 5.2 will explain the performance parameters. Section 5.3 provides details on benchmarking and Section 5.4 gives the experimental results.

5.1. System description

We used NASA projects from the PROMISE repository. Public datasets known as KC2, KC3, CM1, MW1, MC2, and PC1 were applied during the experiments. The KC2 dataset was a data processing project developed with C++ language and included 523 modules, of which 15% were defective. The KC3 dataset was a storage management project developed with Java language and included 458 modules, of which 9% were defective [20]. The CM1 dataset was a NASA spacecraft instrument project developed with C language and included 498 modules, of which 10% were defective. MW1 was a software project related to zero-gravity experiments and was developed with C++ language. It included 434 modules, of which 8% were defective. MC2 was a video guidance system developed with C++ language and included 161 modules, of which 32% were defective. PC1 was flight software for an Earth-orbiting satellite developed with C language and included 1109 modules, of which 7% were defective.

Datasets included 21 metrics that were developed by Halstead [21] and McCabe [22]. As explained in Section 4, derived metrics were discarded because they were redundant data for the datasets we used. Metrics we preferred were consistent with the metrics used by researchers who develop models for defect predictions. Rodriguez et al. [20] used the same metrics set to evaluate the models they developed. They emphasized the negative impact of irrelevant and redundant features in their study. When the number of features increases for a problem, the number of data instances we collect must increase; this problem is known as the curse of dimensionality [20]. A dataset that has fewer features works faster and generally produces better results. Metrics we used belong to three categories. The first category of metrics are McCabe metrics: lines of code, cyclomatic complexity, essential complexity, and design complexity. Cyclomatic complexity shows the number of independent paths in a program that must be tested to ensure the correct behavior of the program. Essential complexity explains how much complexity is left if we remove well-structured complexity; it can be defined as the cyclomatic complexity of the reduced control flow graph after replacing control structures. By using this metric, unstructured constructs are detected and the degree of structuredness is measured. Design complexity measures the interaction between modules. The second category of metrics are Halstead metrics. These metrics are counts of tokens in a source code. For example, logical/arithmetic/relational/assignment operators might be used in a program and variables/constants might be used as operands. Number of operators, number of operands, unique number of operators, and unique number of operands are the metrics that were used from this category. The last category is the branch category. We had only one metric, which was branch count. It shows the number of branches in a flow graph.

5.2. Performance evaluation

In this study, we used AUC, PF, and PD for benchmarking.

5.3. Benchmarking

Our benchmarking used two techniques, naive Bayes with logNum and the threshold-based fault prediction approach, on six NASA datasets. For each dataset, the other five datasets were combined and considered as a training dataset. For the naive Bayes approach, the training dataset and test dataset were filtered with the logNum function. After filtering, training was performed with naive Bayes. After prediction, performance evaluation parameters were calculated. For threshold-based fault prediction, the training dataset was used to calculate threshold values. After the corresponding thresholds were calculated, prediction was done on the test dataset. Performance evaluation parameters were calculated for these two techniques, and then these results were compared.

5.4. Results and analysis

We used AUC, PD, and PF parameters for performance evaluation. Tables 2 and 3 show the results for threshold-based fault prediction approach and naive Bayes with logNum. The results are very similar. Tables show that

PD values of the threshold-based approach are larger than the naive Bayes-based approach. This means that fault-prone modules are detected more efficiently. However, PF values of the threshold-based approach are slightly larger than those of the other approach.

These experimental results show that the naive Bayes-based fault prediction approach and our threshold-based fault prediction approach both work well when a CC dataset is used. Our main purpose was to evaluate the effect of CC datasets when there are no prior fault data and we empirically observed that it helps companies.

Menzies et al. [23] stated that mission-critical systems that are risk-averse may prefer high PF and high PD. For less critical software systems, which are cost-averse, a midrange PD is preferred if the PF value is low. According to this information, we can specify that the suggested threshold-based approach can be preferred for mission-critical systems because we have higher PD and PF does not increase so much. Although our purpose was not to find a better prediction approach than the naive Bayes-based technique, the suggested approach works better for mission-critical systems. We did not perform statistical significance tests because the purpose of this study was not to find a better model for cross-project fault prediction. Instead, the purpose was to understand whether or not the cross-project fault prediction works in the context of missing fault labels.

In Table 2, the average AUC is 68.5% when the threshold-based approach is applied, and in Table 3, it is 68.3% when the naive Bayes-based technique is used. In Table 2, PD is 78.4% for the KC2 dataset and in Table 3, PD is 76.4% for the same dataset. An AUC value close to 1.0 means that the classifier is perfect, and an AUC value close to 0.50 means that the classifier has random prediction performance.

Table 2. Results for the proposed approach.

Dataset	AUC	PD	PF
KC2	0.784	0.790	0.215
KC3	0.695	0.604	0.212
CM1	0.674	0.766	0.421
MW1	0.654	0.645	0.336
MC2	0.620	0.634	0.398
PC1	0.685	0.723	0.352
Average	0.685	0.693	0.322

Table 3. Results for naive Bayes with logNum.

Dataset	AUC	PD	PF
KC2	0.791	0.764	0.181
KC3	0.690	0.581	0.200
CM1	0.668	0.708	0.372
MW1	0.675	0.645	0.295
MC2	0.633	0.596	0.330
PC1	0.645	0.618	0.328
Average	0.683	0.652	0.284

5.5. Threats to validity

Six NASA projects were used as CC datasets in this study. Although these projects were implemented by different companies in different regions of the United States for NASA, they are mostly from the aerospace domain. The impact of other domains on this problem might be different and, therefore, we must emphasize that these CC fault prediction models were evaluated within this domain. Another threat to generalization might be the use of NASA datasets for the software engineering industry. However, it was reported that NASA data are relevant for this industry [5]. Another possible threat is the selection of metrics in our threshold calculation approach. Different researchers might use different metrics and, therefore, the performance might be different than the results we calculated.

6. Conclusions and future work

Most fault prediction models use prior fault data. However, companies do not always have fault data for fault prediction. In this study, we focused on the use of CC data because we observed that, in many cases, previous models do not work well.

Turhan et al. [5] used naive Bayes on CC data, which motivated us to use CC data in the absence of fault data. Instead of machine learning-based CC data usage, we aimed to develop a technique considering the idea of software metrics threshold values.

The proposed threshold-based fault prediction technique achieved a larger PD value than the naive Bayes-based approach. For mission-critical applications, PD values are more important than PF values because all of the faults should be removed before deployment. Therefore, we suggest that companies use the proposed threshold-based approach for mission-critical applications, in addition to the naive Bayes-based approach.

The findings in this study support the findings of Turhan et al. [5] that suggest using CC data. In this study, we provided new evidence that CC data are useful for building fault prediction models. We also proposed a new fault prediction approach.

For future work, we will try to improve the proposed threshold-based approach to achieve smaller PF values and higher PD values. In addition, we will investigate the effect of projects from different domains. In this study, all of the projects were from the aerospace domain, but finding project data from a similar domain is not an easy task. Using projects from different domains and different processes may not lead to acceptable performance and we will evaluate these factors in order to generalize the proposed model.

References

[1] Catal C, Diri, B. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Inf Sci 2009; 179: 1040-1058.

[2] Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE T Software Eng 2008; 34: 485-495.

[3] Catal C, Diri, B. A systematic review of software fault prediction studies. Expert Syst Appl 2009; 36: 7346-7354.

[4] Kitchenham BA, Mendes E, Travassos GH. Cross versus within-company cost estimation studies: a systematic review. IEEE T Software Eng 2007; 33: 316-329.

[5] Turhan B, Menzies T, Bener AB, Stefano JD. On the relative value of cross-company and within-company data for defect prediction. Empir Softw Eng 2009; 14: 540-578.

[6] Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: ESEC/FSE'09; 24–28 August 2009; Amsterdam, the Netherlands. New York, NY, USA: ACM. pp. 91-100.

[7] Shatnawi R, Li W, Swain J, Newman T. Finding software metrics threshold values using roc curves. J Softw Maint-Res Pr 2010; 22: 1-16.

[8] Zhong S, Khoshgoftaar TM, Seliya N. Unsupervised learning for expert-based software quality estimation. In: 8th International Symposium On High Assurance Systems Engineering; 25–26 March 2004; Tampa, FL, USA. New York, NY, USA: IEEE. pp. 149-155.

[9] Catal C, Sevim U, Diri B. Clustering and metrics thresholds based software fault prediction of unlabeled program modules. In: ITNG '09; 27–29 April 2009; Las Vegas, NV, USA. Washington, DC, USA: IEEE. pp. 199-204.

[10] Catal C. Software mining and fault prediction. Wires Data Mining Knowl Discov 2012; 2: 420-426.

[11] Zhong S, Khoshgoftaar TM, Seliya N. Analyzing software measurement data with clustering techniques. IEEE Intell Syst 2004; 19: 20-27.

[12] Seliya N, Khoshgoftaar TM. Software quality analysis of unlabeled program modules with semi-supervised clustering. IEEE T Syst Man Cy A 2007; 37: 201-211.

[13] Seliya N, Khoshgoftaar TM, Zhong S. Semi-supervised learning for software quality estimation. In: IEEE 2004 Conference on Tools with Artificial Intelligence; 15–17 November 2004; Boca Raton, FL, USA. Los Alamitos, CA, USA: IEEE. pp. 183-190.

[14] Catal C, Diri B. Unlabelled extra data do not always mean extra performance for semi-supervised fault prediction. Expert Syst 2009; 5: 458-471.

[15] Catal C, Alan O, Balkan K. Class noise detection based on software metrics and roc curves. Inf Sci 2011; 181: 4867-4877.

[16] Menahem E, Rokach L, Elovici Y. Troika - an improved stacking schema for classification tasks. Inf Sci 2009; 179: 4097-4122.

[17] Zhang M, Peña JM, Robles V. Feature selection for multi-label naive Bayes classification. Inf Sci 2009; 179: 3218-3229.

[18] Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. IEEE T Software Eng 2007; 33: 2-13.

[19] Menzies T, Turhan B, Bener A, Gay G, Cukic B, Jiang Y. Implications of ceiling effects in defect predictors. In: 4th International Workshop on Predictor Models in Software Engineering; 10–18 May 2008; Leipzig, Germany. New York, NY, USA: ACM. pp. 47-54.

[20] Rodriguez D, Ruiz R, Riquelme JC, Aguilar-Ruiz JS. Searching for rules to detect defective modules: a subgroup discovery approach. Inf Sci 2012; 191: 14-32.

[21] Halstead M. Elements of Software Science. New York, NY, USA: Elsevier, 1977.

[22] McCabe T. A complexity measure. IEEE T Software Eng 1976; 2: 308-320.

[23] Menzies T, Milton Z, Turhan B, Cukic B, Jiang Y, Bener A. Defect prediction from static code features: current results, limitations, new approaches. Automat Softw Eng 2010; 17: 375-407.