International Journal of Software Engineering
and Knowledge Engineering
Vol. 27, Nos. 9&10 (2017) 1427–1438

DOI: 10.1142/S0218194017400046

Improving Cross-Company Defect Prediction with Data Filtering

Xiao Yu*, Jin Liu*,†,‡, Weiqiang Peng*
and Xingyu Peng*

*State Key Laboratory of Software Engineering, School of Computer
Wuhan University, Wuhan, Hubei 430072, P. R. China

†Guangxi Key Laboratory of Trusted Software
Guilin University of Electronic Technology, Guangxi, P. R. China
‡jinliu@whu.edu.cn

Defect prediction aims to estimate software reliability via learning from historical defect data. Cross-company defect prediction (CCDP) is a practical way that trains a prediction model by exploiting one or multiple projects of a source company and then applies the model to the target company. Unfortunately, larger irrelevant cross-company (CC) data usually makes it difficult to build a CCDP model with high performance. To address such issues, this paper proposes a data filtering method based on agglomerative clustering (DFAC) for CCDP. First, DFAC combines within-company (WC) instances and CC instances and uses agglomerative clustering algorithm to group these instances. Second, DFAC selects subclusters which consist of at least one WC instance, and collects the CC instances in the selected subclusters into a new CC data. Compared with existing data filter methods, the experiment results from 15 public PROMISE datasets show that DFAC increases the pd value, reduces the pf value and achieves higher G-measure value.

Keywords: Defect prediction; cross-company defect prediction; data filter; agglomerative clustering.

1. Introduction

Defect prediction is one of the most important software quality assurance techniques [1]. Based on the investigation of historical metrics [2], defect prediction aims to detect the defect proneness of new software modules. Therefore, defect prediction is often used to help to reasonably allocate limited development and maintenance resources [3–5]. With the advent of big data era and the development of machine learning techniques [6], many machine learning algorithms are applied to solve practical problems in life [7–9].

‡Corresponding author.

In the same way, many efficient defect prediction methods using statistical methods or machine learning techniques have been proposed [10–21], but they are usually confined to predicting a given software module being faulty or nonfaulty by means of some binary classification techniques. Within-company defect prediction (WCDP) works well if sufficient data is available to train a defect prediction model. However, it is difficult for a new project to perform WCDP if there is limited historical data. Cross-company defect prediction (CCDP) is a practical approach to solve the problem. It trains a prediction model by exploiting one or multiple projects of a source company and then applies the model to target company [22].

In recent years, most existing CCDP approaches have been proposed. Unfortunately, larger irrelevant cross-company (CC) data usually makes it difficult to build a CCDP model with high performance. Therefore, how to weaken the impact of irrelevant CC data to improve the performance of CCDP is a big challenge. The ability to transfer knowledge from a source company to a target company depends on how they are related. The stronger the relationship, the more usable will be CC data. The performance of CCDP is generally poor because of larger irrelevant CC data. Previous work [23] found that using raw CC data directly would increase false alarm rates due to irrelevant instance in CC data, so several data filtering works should be done before building the prediction model. For example, Turhan *et al.* [23] and Peters *et al.* [24] proposed the nearest neighbor (NN) filter and the Peters filter to select the CC instances which are mostly similar to within-company (WC) data as the training dataset.

Considering such challenge, this paper proposes a data filtering method based on agglomerative clustering (DFAC) for CCDP. First, DFAC combines within-company instances and cross-company instances and uses agglomerative clustering algorithm to group these instances. Second, DFAC selects subclusters which consist of at least one WC instance, and collects the CC instances in the selected subclusters into a new CC data. We evaluate our proposed method, DFAC, on 15 publicly available project datasets. Experimental results show that DFAC can effectively filter out the irrelevant CC instances to improve the overall prediction performance. On most of the project datasets under evaluation, DFAC performs well and achieves the best G-measure values.

The remainder of this paper is organized as follows. Section 2 presents the related works. Section 3 describes our proposed DFAC method. Section 4 demonstrates the experimental results. Section 5 discusses the potential threats to validity. Finally, Sec. 6 addresses the conclusion and points out the future work.

2. Related Works

In this section, we first review the existing defect prediction methods. Then, we briefly review the CCDP or cross-project defect prediction (CPDP) methods.

2.1. *Defect prediction*

Many researchers have proposed various models for predicting the module being faulty or nonfaulty in terms of within-project defect prediction (WPDP).

Support vector machine [10], neural networks [11–13] and decision trees [14, 15] paved the way for classification-based methods in the field of defect prediction. These methods used software metrics to properly predict whether a module is defect-prone or not. However, feature irrelevance and imbalanced nature of the defect datasets degraded the prediction performance. Therefore, some feature selection methods [16, 17] and class imbalance learning methods [18] have been proposed to cope with feature irrelevance and class imbalance problem for defect prediction. For example, Wang *et al.* [17] leveraged deep belief network to automatically learn semantic features from source code.

In the process of defect prediction, misclassification of different software defect classes can be divided into two types, namely "Type I" and "Type II". "Type I" misclassification cost and "Type II" misclassification cost are different. Therefore, some cost-sensitive learning methods [19] have been proposed to address this issue by generating a classification model with minimum misclassification cost. In addition, several methods [20, 21] based on ensemble learning have been proposed to address the defect prediction problem. However, these methods are confined to predicting a given software module being faulty or nonfaulty.

2.2. *Cross-company defect prediction*

In order to solve the problem that the new companies have too limited historical data to perform WCDP well, the cross-project and cross-company defect predictions appeared. Zimmermann *et al.* [25] studied CCDP models on 12 real-world applications' datasets. Their results indicated that CCDP is still a serious challenge because of the different distribution between the training project data and the target project data. In order to narrow the distribution gap, there are three mainstream ways.

The first one is to apply the data filtering method to find the best suitable training data (see e.g. [23, 24, 26]). For example, Turhan *et al.* [23] proposed an NN filter to select cross-company data. Peters *et al.* [24] introduced the Peters filter to select training data via the structure of other projects. They compared the filter with two other approaches for quality prediction to assess the performance of the Peters filter, and found that (1) WCDPs are weak for small datasets and (2) the Peters filter + CCDP builds better and more useful predictors. Kawata *et al.* [26] proposed and examined a data filtering method based on one of the advanced clustering algorithms, the density-based spatial clustering (DBSCAN). Our proposed approach DFAC also belongs to this kind.

The second mainstream way is to design effective defect predictor based on transfer learning techniques (see e.g. [22, 27, 28]). For instance, Ma *et al.* [27] proposed a novel algorithm called transfer naive bayes (TNB) to transfer cross-company

data information into the weights of the training data and then build the predictor based on reweighted CC data. The results indicated that TNB is more accurate in terms of the area under the receiver operating characteristic curve (AUC), with less runtime than the state-of-the-art methods and can effectively achieve the CCDP task. Chen *et al.* [28] proposed double transfer boosting (DTB) model. Another challenge in CCDP is that the set of metrics between the source company data and target company data are usually heterogeneous. The heterogeneous CCDP (HCCDP) task is that the source and target company data are heterogeneous. Jing *et al.* [22] provided an effective solution for HCCDP. They proposed a unified metric representation (UMR) for the data of source and target companies and introduced canonical correlation analysis (CCA), an effective transfer learning method, into CCDP to make the data distributions of source and target companies similar. Results showed that their approach significantly outperforms state-of-the-art CCDP methods for HCCDP with partially different metrics and for HCCDP with totally different metrics, their approach is also effective.

The third mainstream way is to apply unsupervised classifier that does not require any training data to perform CCDP (see e.g. [29, 30]), therefore the distribution gap between the training project data and the target project data is no longer an issue. For instance, Zhang *et al.* [30] proposed to apply a connectivity-based unsupervised classifier that is based on spectral clustering to perform CPDP.

3. Methodology

In this section, we first present our proposed approach DFAC. Then, we give an example to illustrate our approach.

3.1. *DFAC*

The main goal of data filter is to select the most valuable training data for the CCDP model by filtering out irrelevant instances in CC data. In this paper, we propose the DFAC algorithm.

DFAC consists of two stages. In the first stage, DFAC combines within-company instances and cross-company instances and uses agglomerative clustering algorithm to group these instances. The main goal of agglomerative clustering is to partition WC instances and CC instances into k clusters such that instances in the same cluster are similar and instances in different clusters are dissimilar to each other.

Agglomerative clustering is an iterative process, which merges current clusters continuously. It is possible that a current cluster only contains one instance, e.g. each instance is treated as one cluster at the beginning of the iteration. In agglomerative clustering, instances are merged into clusters according to the distances between current clusters. We employ the average linkage method to define the distance of two current clusters. The average linkage between two clusters is defined as the average

of the distances between any instance pairs between two clusters. Suppose that U_a and U_b are two current clusters during the clustering process. The distance of the two clusters $D_{a,b}$ can be calculated by the following formula:

$$D_{a,b} = \frac{1}{m_a m_b} \sum_{x_i \in U_a, x_j \in U_b} d_{i,j}, \tag{1}$$

where m_a and m_b are the numbers of instances inside clusters U_a and U_b, respectively, and $d_{i,j}$ is the distance between two instances x_i and x_j. We define the distance $d_{i,j}$ of two instances x_i and x_j with Euclidean distance.

In our work, we choose the number of final clusters k by maximizing the increment of inconsistency coefficient (IC). Inconsistency coefficient is used to quantitatively express the relatively consistent of one link [31]. A *link* denotes an action of merging two current clusters. The value of inconsistency coefficient can be calculated by comparing the distance of the current link and the average distance of its neighbors. The neighbors of one specific link denote all children links that lead to this link as well as the link itself. For each link, we count its IC value of a link L_{curr} to measure the change of clustering as follows:

$$\text{IC}(L_{\text{curr}}) = \frac{D_{L_{curr}} - \text{avg}(D_{L_{\text{neighbor}}})}{\text{Std}(D_{L_{\text{neighbor}}})}, \tag{2}$$

where $\text{Std}(D_{L_{\text{neighbor}}})$ denotes the standard deviation of all links in its neighbors. We define the increment of IC values between two links as $\Delta_{L_{\text{curr}}, L_{\text{prev}}} = \text{IC}(L_{\text{curr}}) - \text{IC}(L_{\text{prev}})$, where L_{curr} and L_{prev} denote a current link and its previous link, respectively. Then we find the link with the maximal increment value and stop the clustering process before this link.

DFAC assumes that the CC instances in the same cluster as WC instances are the most valuable instances in CC data. Therefore, in the second stage, DFAC selects subclusters which consist of at least one WC instance, and collects the CC instances in the selected subclusters into a new CC data.

3.2. *Example*

Figure 1 shows the resulting clusters of a set of instances using the agglomerative clustering algorithm, where "◯" represents the CC instance and "△" represents the WC instance.

These instances are partitioned into three clusters by using the agglomerative clustering algorithm, namely $C_1 = \{x_1, x_4, x_5, x_6, x_{17}, x_{21}\}$, $C_2 = \{x_2, x_3, x_9, x_{15}, x_{22}\}$ and $C_3 = \{x_7, x_8, x_{12}, x_{13}, x_{14}, x_{16}, x_{18}, x_{20}\}$. Take the cluster C_1 for example, since the CC instances x_1, x_4, x_5 and x_6 are in the same cluster as the WC instances x_{17} and x_{21}, these CC instances are selected to form the final CC training data. Take the cluster C_2 for example, since the cluster does not consist of any WC instance, the CC instances in this cluster are discarded. The CC instances in the cluster C_3 are

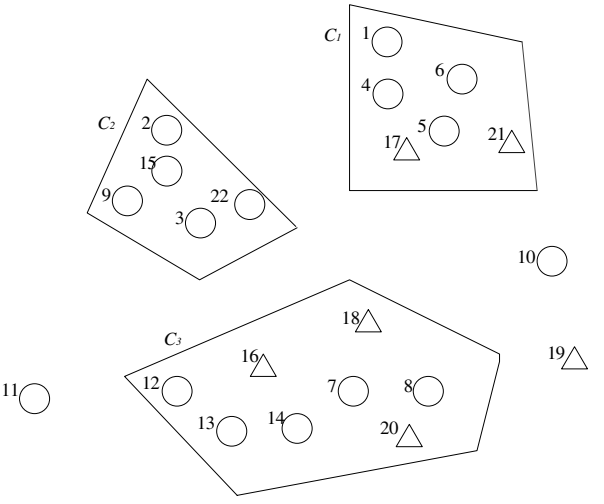

Fig. 1. Resulting clusters of a set of instances using agglomerative clustering algorithm.

selected in the same manner. Therefore, the final CC training instances include x_1, x_4, x_5, x_6, x_7, x_8, x_{12}, x_{13} and x_{14}.

4. Experiments

In this section, we evaluate our DFAC method to perform CCDP empirically. We first introduce the experiment datasets, the performance measures and the experimental procedure. Then, in order to investigate the performance of DFAC, we perform some empirical experiments.

4.1. *Experiment datasets*

In this experiment, we employ 15 available and commonly used datasets which can be obtained from PROMISE [32]. The 15 datasets have the same 20 attributes, so we can apply all attribute information directly. Table 1 tabulates the details about the datasets, and Table 2 shows a more detailed description of the 20 independent code attributes.

Table 1. Details of experiment datasets.

Project	Examples	% Defective	Description
ant	125	16	Open-source
arc	234	11.5	Academic
camel	339	3.8	Open-source

Table 1. (*Continued*)

Project	Examples	% Defective	Description
elearn	64	7.8	Academic
jedit	272	33.1	Open-source
log4j	135	25.2	Open-source
lucene	195	46.7	Open-source
poi	237	59.5	Open-source
prop-6	660	10	Proprietary
redactor	176	15.3	Academic
synapse	157	10.2	Open-source
system	65	13.8	Open-source
tomcat	858	9	Open-source
xalan	723	15.2	Open-source
xerces	162	47.5	Open-source

Table 2. Code attributes of the datasets.

No.	Attribute	Description
1	wmc	Weighted methods per class
2	dit	Depth of inheritance tree
3	noc	Number of children
4	cbo	Coupling between object classes
5	rfc	Response for a class
6	lcom	Lack of cohesion in methods
7	ca	Afferent couplings
8	ce	Efferent couplings
9	npm	Number of public methods
10	lcom3	Lack of cohesion in methods
11	loc	Lines of code
12	dam	Data access metric
13	moa	Measure of aggregation
14	mfa	Measure of functional abstraction
15	cam	Cohesion among methods of class
16	ic	Inheritance coupling
17	cbm	Coupling between methods
18	amc	Average method complexity
19	max_cc	Maximum McCabe's cyclomatic complexity
20	avg_cc	Average McCabe's cyclomatic complexity

4.2. *Performance measures*

In the experiment, we employ three commonly used performance measures including pd, pf and G-measure. They are defined in Table 3 and summarized as follows.

Probability of detection or *pd* is the measure of defective modules that are correctly predicted within the defective class. The higher the *pd*, the fewer the false negative results.

Probability of false alarm or *pf* is the measure of nondefective modules that are incorrectly predicted within the nondefective class. Unlike *pd*, the lower the *pf* value, the better the results.

Table 3. Performance measures.

		Actual	
		Yes	No
Predicted	Yes	TP	FP
	No	FN	TN
pd		$\frac{TP}{TP+FN}$	
pf		$\frac{FP}{FP+TN}$	
G-measure		$\frac{2*pd*(1-pf)}{pd+(1-pf)}$	

G-measure is a trade-off measure that balances the performance between *pd* and *pf*. A good prediction model should have high *pd* and low *pf*, and thus leading to a high *G-measure*.

4.3. *Experimental procedure*

In order to confirm whether DFAC can perform better than other data filtering methods, we compare DFAC with three state-of-the-art data filtering approaches. More details are provided below.

NN filter [23] is based on the widely used classification method K-nearest neighbors (KNN) algorithm to filter irrelevant CC data. It can find out the most similar $K \times N$ instances from CC data where N is the number of instances in WC data and K is the parameter of the KNN method. In our experiment, we choose K as 10.

Peters filter [24] first divides the target project data and the cross-project data into subclusters using the K-means algorithm and then selects subclusters which have at least one record of the target project data. From each selected subcluster, Peters filter finally finds the 10 nearest neighbors of each record of the cross-project data from the target project data in the same cluster, and selects the nearest neighbor of each record of the selected target project data from the cross-project data which selects that target record as its nearest neighbor.

DBSCAN filter [26] is based on the DBSCAN algorithm. DBSCAN defines the high density with two parameters: the distance which determines whether two records are close to one another and the number of records which determines whether a core sample is in a dense area. In our experiment, we choose the two parameters as 10 and 10, respectively.

In every experiment, one dataset is selected as WC data and the rest are regarded as CC data to conduct the experiment. The CC data is considered as basic training data which will be adjusted in every experiment. All data filtering methods are done on CC data. Then processed CC data are used to build the CCDP model. Finally, the resulting model is evaluated on the WC data. In this experiment, we choose naive Bayes (NB) [33] as the CCDP model due to its effectiveness in defects prediction.

4.4. *Experiment results*

The comparison results of 15 projects with NB classifier based on pd, pf and *G*-measure are summarized in Table 4. Table 4 shows that DFAC performs well with better average pd and pf values than all the other data filtering methods. It shows that the NN filter often achieves good pd but bad pf so that it usually ends up with low *G*-measure value. The Peters filter achieves the best pd but the worst pf. The performance of DBSCAN filter sometimes seems to have lower pf value than the NN filter but it has lower pd value. It is very clear that DFAC has lower pf value and higher pd value than the other three data filtering methods. In the aspect of pd value, the DFAC approach increases the pd value to an extent on most datasets. The pd values of eight projects are better than others. On most tests, DFAC achieves higher *G*-measure than the other data filtering methods. In total, DFAC has acceptable pf values and can obtain better pd values in most experiments we conducted, and it almost always achieves the higher *G*-measure value than other data filtering methods. In other words, DFAC outperforms other data filtering methods, therefore it can be an effective data filtering method.

Table 4. The pd, pf and *G*-measure values from 15 projects.

Test data	NN filter			Peters filter			DBSCAN filter			DFAC		
	pd	pf	*G*-measure	pd	pf	*G*-measure	pd	pf	*G*-measure	pd	pf	*G*-measure
ant	0.829	0.537	0.594	0.752	0.450	0.635	0.768	0.539	0.576	0.824	0.454	0.657
arc	0.829	0.593	0.546	0.831	0.481	0.639	0.824	0.594	0.544	0.838	0.592	0.549
camel	0.889	0.879	0.213	0.880	0.462	0.668	0.875	0.792	0.336	0.912	0.701	0.450
elearn	0.842	0.705	0.437	0.695	0.600	0.508	0.860	0.704	0.440	0.877	0.703	0.444
jedit	0.779	0.337	0.716	0.918	0.689	0.465	0.775	0.345	0.710	0.783	0.329	0.723
log4j	0.860	0.342	0.746	0.881	0.706	0.441	0.835	0.374	0.716	0.853	0.361	0.731
lucene	0.657	0.374	0.641	0.904	0.604	0.550	0.646	0.383	0.631	0.651	0.378	0.636
poi	0.545	0.338	0.598	0.969	0.759	0.386	0.521	0.354	0.577	0.531	0.352	0.584
prop-6	0.803	0.531	0.592	0.818	0.606	0.532	0.801	0.370	0.705	0.791	0.521	0.597
redactor	0.747	0.372	0.682	0.872	0.815	0.306	0.759	0.338	0.707	0.759	0.338	0.707
synapse	0.759	0.508	0.597	0.660	0.188	0.728	0.773	0.438	0.651	0.788	0.363	0.705
system	0.828	0.485	0.635	0.804	0.556	0.572	0.828	0.485	0.635	0.828	0.485	0.635
tomcat	0.854	0.591	0.553	0.894	0.545	0.603	0.850	0.577	0.565	0.853	0.601	0.544
xalan	0.820	0.565	0.568	0.804	0.527	0.595	0.817	0.574	0.560	0.827	0.552	0.581
xerces	0.579	0.425	0.577	0.929	0.870	0.228	0.566	0.439	0.563	0.566	0.439	0.563
Avg	0.775	0.505	0.580	0.841	0.591	0.524	0.767	0.487	0.594	0.778	0.480	0.606

Figure 2 shows the box-plots of *G*-measure values for NN filter, Peters filter, DBSCAN filter and DFAC, from 15 projects with NB classifier. It seems that Peters filter approach always has low *G*-measure compared to NN filter, DBSCAN filter and DFAC. Although DFAC has the same minimum with DBSCAN filter, the median value of DFAC is higher than those of NN filter, Peters filter and DBSCAN filter.

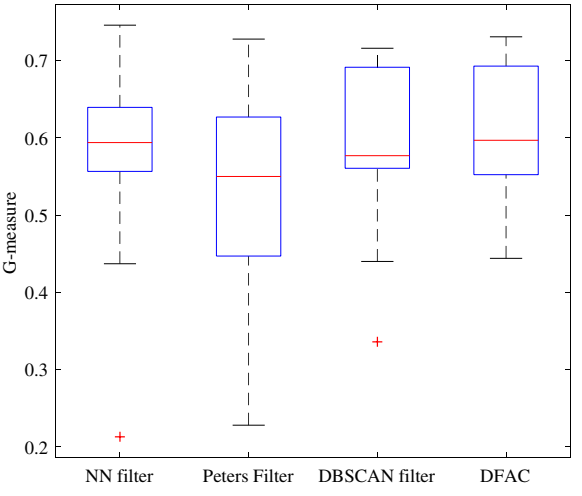

Fig. 2. Box-plots of G-measures from 15 projects.

5. Threats to Validity

In this section, we discuss several validity threats that may have an impact on the results of our studies.

External validity. Threats to external validity occur when the results of our experiments cannot be generalized. As a preliminary result, we performed our experiments on the 15 datasets to explore the generality of our method. Although these datasets have been widely used in many software defect prediction studies, we still cannot claim that our method can be generalized to other datasets. Nevertheless, this work provides a detailed experimental description, including parameter settings (default parameter settings specified by sklearn), thus other researchers can easily replicate our method on new datasets.

Internal validity. We compared our method with NN filter and DBSCAN filter. To avoid the potential faults as much as possible during the implementation process of the experiment, we implement these models based on the Python machine learning library sklearn.

Construct validity. Threats to construct validity focus on the bias of the measures used to evaluate the performance of CCDP. In our experiments, we mainly use pd, pf and G-measure to measure the effectiveness of the four approaches. Nonetheless, other evaluation measures such as AUC measure can also be considered.

6. Conclusion and Future Work

In this paper, we propose a data filtering method based on agglomerative clustering for CCDP. The method consists of two stages. In the first stage, DFAC combines WC instances and CC instances and uses agglomerative clustering algorithms to

group these instances. In the second stage, DFAC selects subclusters which consist of at least one WC instance, and collects the CC instances in the selected subclusters into a new CC data. We conduct experiments on the 15 datasets to evaluate the performance of the proposed DFAC method. The experimental results indicate that the proposed method can effectively filter out and weaken the impact of irrelevant data to improve the performance of CCDP. The proposed DFAC method is an effective data filtering method for CCDP.

In future, we would like to employ more project datasets to validate the generalization of our proposed method.

Acknowledgments

The authors would like to acknowledge the support provided by the grants of the National Natural Science Foundation of China (Nos. 61572374, U1636220, 61472423), Guangxi Key Laboratory of Trusted Software (No. kx201607), Academic Team Building Plan for Young Scholars from Wuhan University (WHU2016012).

References

1. H. Qing *et al.*, Cross-project software defect prediction using feature-based transfer learning, in *Proc. 7th Asia-Pacific Symp. Internetware*, 2015, pp. 74–82.
2. K. Gao *et al.*, Choosing software metrics for defect prediction: An investigation on feature selection techniques, *Soft. — Pract. Exp.* **41**(5) (2011) 579–606.
3. M. Shepperd, D. Bowes and T. Hall, Researcher bias: The use of machine learning in software defect prediction, *IEEE Trans. Softw. Eng.* **40**(6) (2014) 603–616.
4. Q. Song *et al.*, A general software defect proneness prediction framework, *IEEE Trans. Softw. Eng.* **37**(3) (2011) 356–370.
5. X. Yang, K. Tang and X. Yao, A learning-to-rank approach to software defect prediction, *IEEE Trans. Reliab.* **64**(1) (2015) 234–246.
6. Z. Liu *et al.*, A task-centric cooperative sensing scheme for mobile crowdsourcing systems, *Sensors* **16**(5) (2016) 746.
7. Z. Xu *et al.*, Hierarchy-cutting model based association semantic for analyzing domain topic on the web, *IEEE Trans. Ind. Inf.* **13**(4) (2017) 1941–1950.
8. Z. Xu *et al.*, Multi-modal description of public security events using surveillance and social data, *IEEE Trans. Big Data*, 2017.
9. Z. Liu *et al.*, UCOR: An unequally clustering-based hierarchical opportunistic routing protocol for WSNs, in *Proc. Int. Conf. Wireless Algorithms, Systems, and Applications*, 2013, pp. 175–185.
10. Z. Yan, X. Chen and P. Guo, Software defect prediction using fuzzy support vector regression, in *Proc. Int. Symp. Neural Networks*, 2010, pp. 17–24.
11. Ö. F. Arar and K. Ayan, Software defect prediction using cost-sensitive neural network, *Appl. Soft Comput.* **33** (2015) 263–277.
12. V. Vashisht *et al.*, A framework for software defect prediction using neural networks, *J. Softw. Eng. Appl.* **8**(8) (2015) 384–394.
13. E. Erturk and E. A. Sezer, Iterative software fault prediction with a hybrid approach, *Appl. Soft Comput.* **49** (2016) 1020–1033.

14. J. Wang, B. Shen and Y. Chen, Compressed C4.5 models for software defect prediction, in *Proc. 12th Int. Conf. Quality Software*, 2012, pp. 13–16.
15. N. Seliya and T. M. Khoshgoftaar, The use of decision trees for cost-sensitive classification: An empirical study in software quality prediction, *Data Min. Knowl. Discov.* **1**(5) (2011) 448–459.
16. Z. Xu *et al.*, MICHAC: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering, in *Proc. 23rd IEEE Int. Conf. Software Analysis, Evolution, and Reengineering*, Vol. 1, 2016, pp. 370–381.
17. S. Wang, T. Liu and L. Tan, Automatically learning semantic features for defect prediction, in *Proc. 38th Int. Conf. Software Engineering*, 2016, pp. 297–308.
18. S. Wang and X. Yao, Using class imbalance learning for software defect prediction, *IEEE Trans. Reliab.* **62**(2) (2013) 434–443.
19. X. Y. Jing, S. Ying, Z. W. Zhang, S. S. Wu and J. Liu, Dictionary learning based software defect prediction, in *Proc. 36th Int. Conf. Software Engineering*, 2014, pp. 414–423.
20. I. H. Laradji, M. Alshayeb and L. Ghouti, Software defect prediction using ensemble learning on selected features, *Inf. Softw. Technol.* **58** (2015) 388–402.
21. M. J. Siers and M. Z. Islam, Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem, *Inf. Syst.* **51** (2015) 62–71.
22. X. Jing *et al.*, Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning, in *Proc. 10th Joint Meeting Foundations of Software Engineering*, 2015, pp. 496–507.
23. B. Turhan, T. Menzies, A. B. Bener and J. Di Stefano, On the relative value of cross company and within-company data for defect prediction, *Empir. Softw. Eng.* **14**(5) (2009) 540–578.
24. F. Peters, T. Menzies and A. Marcus, Better cross company defect prediction, in *Proc. 10th Int. Workshop Mining Software Repositories*, 2013, pp. 409–418.
25. T. Zimmermann, N. Nagappan, H. Gall, E. Giger and B. Murphy, Cross-project defect prediction: A large scale experiment on data vs. domain vs. process, in *Proc. 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. Foundations of Software Engineering*, 2009, pp. 91–100.
26. K. Kawata, S. Amasaki and T. Yokogawa, Improving relevancy filter methods for cross-project defect prediction, in *Applied Computing & Information Technology*, 2016, pp. 1–12.
27. Y. Ma, G. Luo, X. Zeng and A. Chen, Transfer learning for cross-company software defect prediction, *Inf. Softw. Technol.* **54**(3) (2012) 248–256.
28. L. Chen *et al.*, Negative samples reduction in cross-company software defects prediction, *Inf. Softw. Technol.* **62** (2015) 67–77.
29. P. S. Bishnu and V. Bhattacherjee, Software fault prediction using quad tree-based k-means clustering algorithm, *IEEE Trans. Knowl. Data Eng.* **24**(6) (2012) 1146–1150.
30. F. Zhang *et al.*, Cross-project defect prediction using a connectivity-based unsupervised classifier, in *Proc. 38th Int. Conf. Software Engineering*, 2016, pp. 309–320.
31. D. Cordes *et al.*, Hierarchical clustering to measure connectivity in fMRI resting-state data, *Magn. Reson. Imaging* **20**(4) (2002) 3050–317.
32. G. Boetticher, T. Menzies and T. Ostrand, The PROMISE repository of empirical software engineering data (2007), http://promisedata.org/repository.
33. D. D. Lewis, Naive (Bayes) at forty: the independence assumption in information retrieval, in *Proc. European Conf. Machine Learning*, 1998, pp. 4–15.