A Data Filtering Method Based on Agglomerative Clustering

Xiao Yu
State Key Lab. of Software Engineering,
Computer School, Wuhan University,
Wuhan, China
xiaoyu_whu@yahoo.com

Jiansheng Zhang
School of Computer Science and Information Engineering,
HuBei University,
Wuhan, China
jianjian_zhang@qq.com

Peipei Zhou
School of Computer Science and Information Engineering,
HuBei University,
Wuhan, China

Jin Liu*
State Key Lab. of Software Engineering,
Computer School, Wuhan University,
Wuhan, China
Corresponding author email: jinliu@whu.edu.cn

***Abstract*—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 target company. Unfortunately, larger irrelevant cross-company (CC) data usually makes it difficult to build a cross-company defect prediction model with high performance. To address such issues, this paper proposes a data filtering method based on Agglomerative Clustering (DFAC) for cross-company defect prediction. First, DFAC combines within-company instances and cross-company instances and uses Agglomerative clustering algorithms to group these instances. Second, DFAC selects sub-clusters which consist at least one WC instance, and collects the CC instances in the selected sub-clusters into a new CC data. Compared with existing data filter methods, the experimental results on 15 public PROMISE datasets show that DFAC increases PD value, reduces PF value and achieves higher G-measure and AUC values.**

Keywords—software defect prediction;cross-company defect prediction;data filter; Agglomerative clustering

I. INTRODUCTION

Software defect prediction is one of the most important software quality assurance techniques. Based on the investigation of historical metrics [1-2], 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 the practical problems in life [7-9].

DOI reference number: 10.18293/SEKE2017-043

In the same way, many efficient software 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 non-faulty by means of some binary classification techniques. WPDP works well if sufficient data is available to train a defect prediction model. However, it is difficult for a new project to perform WPDP 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 cross-company defect prediction 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 NN filter and the Perters filter to select the CC instances which are mostly similar to WC data as the training dataset.

Considering such challenge, this paper proposes a data filtering method based on Agglomerative clustering (DFAC) for cross-company defect prediction. First, DFAC combines within-company instances and cross-company instances and

uses Agglomerative clustering algorithms to group these instances. Second, DFAC selects sub-clusters which consist at least one WC instance, and collects the CC instances in the selected sub-clusters 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 project datasets under evaluation, DFAC performs the best G-measure values.

The remainder of this paper is organized as follows. Section II presents the related work. Section III describes our proposed DFAC method. Section IV demonstrates the experimental results. Section V discusses the potential threats to validity. Finally, Section VI addresses the conclusion and points out the future work.

II. Related Work

In this section, we first review the existing defect prediction methods. Then, we briefly review the cross-company defect prediction or cross-project defect prediction.

A. Defect prediction

Many researchers have proposed various models for predicting the module being faulty or non-faulty 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 flied 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 software 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, misclassify 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 non-faulty.

B. 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 prediction 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 (e.g., [23, 24, 26]). For example, Turhan et al. [23] proposed a nearest neighbor (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) WCDP are weak for small data sets; 2) the Peters filter + CCDP builds better and more useful predictors.

The second mainstream way is to design effective defect predictor based on transfer learning techniques (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 re-weighted CC data. The results indicated that TNB is more accurate in terms of AUC, within 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 is usually heterogeneous. The heterogeneous CCDP (HCCDP) task is that the source and target company data is 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 (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 Agglomerative clustering to perform CPDP.

III. Methodology

A. DFAC

Previous work [23] found that using raw CC data directly would increase false alarm rates due to irrelevant instances in CC data, so several data filtering works should be done before building the prediction model. 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 a Data Filtering method based on Agglomerative Clustering algorithm (DFAC).

DFAC consists of two stages. In the first stage, DFAC combines within-company instances and cross-company instances and uses Agglomerative clustering algorithms 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 distance between any instance pair 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:

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

where $m_a$and m_b are the number of instances inside clusters U_a and U_b 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.

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

B. Example

Fig.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.

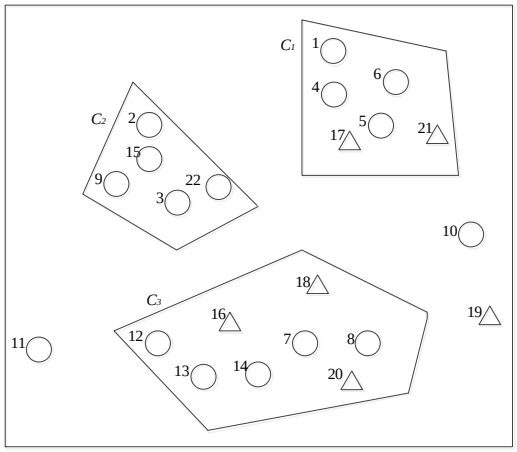

Fig. 1. Resulting clusters of a set of instances using Agglomerative clustering algorithm

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 instance 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 do not consist any WC instance, the CC instances in this cluster are discarded. The CC instances in the cluster C_3 are selected in the same manner. Therefore, the final CC training instances consist of x_1, x_4, x_5, x_6, x_7, x_8, x_{12}, x_{13} and x_{14}.

IV. Experiments

In this section, we evaluate our DFAC method to perform CCDP empirically. We first introduce the experiment dataset, the performance measures and the experimental procedure.

A. Data set

In this experiment, we employ 15 available and commonly used datasets which can be obtained from PROMISE [31]. The 15 datasets have the same 20 attributes, so we can apply all attribute information directly. Table I tabulates the details about the datasets, and Table II shows a more detailed description of the 20 independent code attributes.

TABLE I. Details of Experiment Dataset

Project	***Examples***	***%Defective***	***Description***
ant	125	16	Open-source
arc	234	11.5	Academic
camel	339	3.8	Open-source
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	660	10	Proprietary
redaktor	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 II. 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

B. Performance measures

In the experiment, we employ three commonly used performance measures including *pd*, *pf* and *g-measure*. They are defined in Table 2 and summarized as follows.

TABLE III. PERFORMANCE MEASURES

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

- 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 non-defective modules that are incorrectly predicted within the non-defective class. Unlike pd, the lower the pf value, the better the results.

- 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.

C. Experimental Procedure

In order to confirm whether DFAC can perform better than other data filtering methods, we compare DFAC with four state-of-the-art CCDP 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 while 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.

- **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 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. The procedure will be repeated 30 times in every experiment to avoid sample bias. Then, the mean values of performance are calculated. In this experiment, we choose Naive Bayes (NB) [32] as the CCDP model due to its effectiveness in defects prediction.

D. Experiment results

The comparison results of 15 projects with NB classifier on pd, pf and G-measure are summarized in Table IV. Table IV shows that DFAC performs better average pd and pf values than all the other data filtering methods. It shows that the NN filter often achieves good pd but the worst pf so that it usually ends up with low g-measure value. The performance of DBSCAN filter seems sometimes have lower pf value than the NN filter but it has lower pd value. It's very clear that DFAC has lower pf value and higher pd value than the other two data filtering methods. In the aspect of pd value, the DFAC approach increases the pd value to an extent on most data sets. The pd values of 8 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 value and can obtain better pd values in most experiments we conducted, and it almost always achieve the higher G-measure value than other data filtering methods. In other words, the DFAC approach outperforms other methods, therefore it can be an effective data filtering methods.

Fig. 2 shows the box-plots of G-measure values, for NN filter, DBSCAN filter and DFAC, with NB classifier on 15 projects. It seems that NN filter approach always has low G-measure comparing to DBSACN filter and DFAC. Although DFAC has the same minimum with DBSCAN filter, the median values of both metrics of DFAC are higher than those of NN filter and DBSCAN filter. Meanwhile, the maximum by DFAC is higher than that by DBSCAN filter, but a litter lower than that by NN filter.

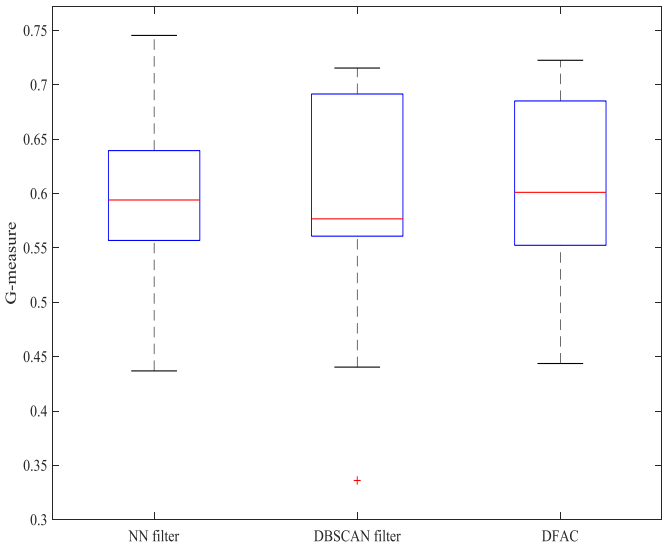

Fig. 2. Box-plots for G-measure on 15 projects.

In total, DFAC has acceptable pd value and can obtain better pf values in most experiments we conducted, and it almost always achieve the higher G-measure than other data filtering methods. In other words, DFAC outperforms other data filtering methods.

TABLE IV. PD, PF AND G-MEASURE VALUES ON 15 PROJECTS

No.	Test data	NN filter			DBSCAN filter			DFAC		
		PD	PF	G	PD	PF	G	PD	PF	G
1	ant	**0.829**	0.537	0.594	0.768	**0.539**	0.576	0.813	0.485	**0.631**
2	arc	0.829	0.593	0.546	0.824	**0.594**	0.544	**0.838**	0.592	**0.549**
3	camel	0.889	**0.879**	0.213	0.875	0.792	0.336	**0.902**	0.704	**0.446**
4	elearn	0.842	**0.705**	0.437	0.860	0.704	0.440	**0.877**	0.703	**0.444**
5	jedit	0.779	0.337	0.716	0.775	**0.345**	0.710	**0.783**	0.329	**0.723**
6	log4j	**0.860**	0.342	**0.746**	0.835	**0.374**	0.716	0.843	0.372	0.720
7	lucene	**0.657**	0.374	**0.641**	0.646	**0.383**	0.631	0.651	0.378	0.636
8	poi	**0.545**	0.338	**0.598**	0.521	**0.354**	0.577	0.531	0.352	0.584
9	prop-6	**0.803**	**0.531**	0.592	0.801	0.370	**0.705**	0.796	0.517	0.601
10	redactor	0.747	**0.372**	0.682	**0.759**	0.338	**0.707**	**0.759**	0.338	**0.707**
11	synapse	0.759	**0.508**	0.597	0.773	0.438	0.651	**0.787**	0.367	**0.702**
12	system	**0.828**	**0.485**	**0.635**	**0.828**	**0.485**	**0.635**	**0.828**	**0.485**	**0.635**
13	tomcat	0.854	0.591	0.553	0.850	0.577	**0.565**	**0.858**	**0.604**	0.542
14	xalan	**0.820**	0.565	**0.568**	0.817	**0.574**	0.560	0.818	0.565	**0.568**
15	xerces	**0.579**	0.425	**0.577**	0.566	**0.439**	0.563	0.566	**0.439**	0.563
	Average	0.775	0.505	0.580	0.767	0.487	0.594	**0.777**	**0.482**	**0.603**

V. 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 detail experimental description, including parameter settings (default parameter settings specified by *sklearn*), thus other researchers can easily replicate our method on new datasets.

Internal validity. In our study, we repeat 30 times to avoid sample bias, and calculate average results to verify the performance of all test methods. We compared our method with NN filter and DBSCAN filter. To avoid the potential faults 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, G-measure to measure the effectiveness of the four approaches. Nonetheless, other evaluation measures such as AUC measure can also be considered.

VI. 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 instance. In the second stage, DFAC selects sub-clusters which consist at least one WC instance, and collects the CC instances in the selected sub-clusters 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 the future, we would like to employ more project datasets to validate the generalization of our proposed method. In addition, we plan to apply our method to a real-life application [33-34].

ACKNOWLEDGMENT

This work is partly supported by the grants of National Natural Science Foundation of China (No.61572374, No.U163620068, No.U1135005) and the Academic Team Building Plan from Wuhan University and National Science Foundation (NSF) (No. DGE-1522883).

REFERENCES

[1] F. Rahman, D. Posnett, P. Devanbu , “Recalling the imprecision of cross-project defect prediction,” Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering,2012, 61.

[2] K. Gao, T.M. Khoshgoftaar, H. Wang, et al, “Choosing software metrics for defect prediction: an investigation on feature selection techniques ,” Software Practice & Experience, 2011,41(5):579-606.

[3] M. Shepperd, D. Bowes, T. Hall, “Researcher Bias: The Use of Machine Learning in Software Defect Prediction,” IEEE Transactions on Software Engineering, 2014,40(6):603-616.

[4] Q. Song, Z. Jia, M. Shepperd, et al, “A general software defect proneness prediction framework,” Software Engineering, IEEE Transactions on, 2011, 37(3): 356-370.

[5] X. Yang, K. Tang, X. Yao, “A Learning-to-Rank Approach to Software Defect Prediction,” IEEE Transactions on Reliability, 2015,64(1): 234-246.

[6] Xu Z, Liu Y, Mei L, et al. “Semantic based representing and organizing surveillance big data using video structural description technology,” Journal of Systems and Software, 2015, 102: 217-225.

[7] Xu Z, Zhang S, Choo K K, et al. “Hierarchy-cutting model based association semantic for analyzing domain topic on the web” IEEE Transactions on Industrial Informatics, 2017.

[8] Xu Z, Mei L, Lu Z, et al. “Multi-modal Description of Public Security Events using Surveillance and Social Data” IEEE Transactions on Big Data, 2017.

[9] Xu Z, Liu Y, Mei L, et al. “The mobile media based emergency management of web events influence in cyber-physical space” Wireless Personal Communications, 2016: 1-14.

[10] Z. Yan, X. Chen, P. Guo, Software defect prediction using fuzzy support vector regression, International Symposium on Neural Networks. Springer Berlin Heidelberg, 2010, pp. 17-24.

[11] Arar, Ömer Faruk, Kürşat Ayan, Software defect prediction using cost-sensitive neural network. Applied Soft Computing 33 (2015) 263-277.

[12] V. Vashisht, M. Lal, G. S. Sureshchandar, et al, A framework for software defect prediction using neural networks. Journal of Software Engineering and Applications, 8.8 (2015) 384.

[13] Erturk, Ezgi, Ebru Akcapinar Sezer, Iterative software fault prediction with a hybrid approach. Applied Soft Computing, 49 (2016) 1020-1033.

[14] J. Wang, B. Shen, Y.Chen, Compressed C4. 5 models for software defect prediction,in: 12th International Conference on Quality Software, 2012, pp. 13-16.

[15] N. Seliya, T. M..Khoshgoftaar, The use of decision trees for cost‐sensitive classification: an empirical study in software quality prediction. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1.5 (2011) 448-459.

[16] Z. Xu, J. Xuan, J. Liu, et al, “MICHAC: Defect Prediction via Feature Selection based on Maximal Information Coefficient with Hierarchical Agglomerative Clustering,” 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2016, 1: 370-381.

[17] S. Wang, T. Liu, L. Tan, “Automatically learning semantic features for defect prediction,” Proceedings of the 38th International Conference on Software Engineering. ACM, 2016: 297-308.

[18] S. Wang, X. Yao, “Using class imbalance learning for software defect prediction,” IEEE Transactions on Reliability,2013,62(2):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. of the 36th International Conference on Software Engineering, 2014, pp. 414-423.

[20] I. H. Laradji, M. Alshayeb, L. Ghouti, Software defect prediction using ensemble learning on selected features, Inform. Softw. Technol. 58 (2015) 388-402.

[21] M. J. Siers, M. Z. Islam, Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem, Information Systems 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. of the 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 496-507.

[23] B. Turhan, T. Menzies, A.B. Bener, J. Di Stefano, “On the relative value of cross company and within-company data for defect prediction,” Empirical Softw. Eng. 2009,14 (5): 540–578.

[24] Peters F, Menzies T, Marcus A, “Better cross company defect prediction,” In: Proc. of the 10th International Workshop on Mining Software Repositories, San Francisco, CA, 2013, 409-418.

[25] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, “Cross-project defect prediction: a large scale experiment on data vs. domain vs. process,” in: Proc. of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM, 2009, pp. 91–100.

[26] K. Kawata, S. Amasaki, T. Yokogawa, “Improving Relevancy Filter Methods for Cross-Project Defect Prediction,” Applied Computing & Information Technology. Springer International Publishing, 2016: 1-12.

[27] Y. Ma, G. Luo, X. Zeng, A. Chen, “Transfer learning for cross-company software defect prediction,” Inform. Softw. Technol. 2012,54 (3): 248–256.

[28] Chen L, Fang B, Shang Z, et al. “Negative samples reduction in cross-company software defects prediction,” Information and Software Technology, 2015, 62: 67-77.

[29] P. S. Bishnu, V. Bhattacherjee, “Software fault prediction using quad tree-based k-means clustering algorithm,” IEEE Transactions on knowledge and data engineering, 2012, 24(6): 1146-1150.

[30] F. Zhang, Q. Zheng, Y. Zou, et al, “Cross-project defect prediction using a connectivity-based unsupervised classifier,” Proceedings of the 38th International Conference on Software Engineering. ACM, 2016: 309-320.

[31] G. Boetticher, T. Menzies, T. Ostrand, The PROMISE Repository of Empirical Software Engineering Data, 2007 <http://promisedata.org/repository>.

[32] D. D. Lewis, Naive (Bayes) at forty the independence assumption in information retrieval, in: European conference on machine learning, 1998, pp. 4–15.

[33] Liu Z, Wei C, Ma Y, et al. UCOR: an unequally clustering-based hierarchical opportunistic routing protocol for WSNs, International Conference on Wireless Algorithms, Systems, and Applications. Springer Berlin Heidelberg, 2013: 175-185.

[34] Liu Z, Wei C, Ma Y, et al. UCOR: an unequally clustering-based hierarchical opportunistic routing protocol for WSNs, International Conference on Wireless Algorithms, Systems, and Applications. Springer Berlin Heidelberg, 2013: 175-185.