Ni C, Liu WS, Chen X *et al.* A cluster based feature selection method for cross-project software defect prediction. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 32(6): 1090–1107 Nov. 2017. DOI 10.1007/s11390-017-1785-0

A Cluster Based Feature Selection Method for Cross-Project Software Defect Prediction

Chao Ni[1], *Student Member*, *IEEE*, Wang-Shu Liu[1], Xiang Chen[1,2], *Senior Member*, *CCF*
Qing Gu[1,*], *Senior Member*, *CCF*, Dao-Xu Chen[1], *Fellow*, *CCF*, *Member*, *ACM*, *IEEE*
and Qi-Guo Huang[1], *Member*, *CCF*

[1] *State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China*
[2] *School of Computer Science and Technology, Nantong University, Nantong 226019, China*

E-mail: {jacknichao920209, liuws0707}@gmail.com; xchencs@ntu.edu.cn; {guq, cdx}@nju.edu.cn
huangqiguo2003@126.com

Received April 21, 2017; revised September 27, 2017.

Abstract Cross-project defect prediction (CPDP) uses the labeled data from external source software projects to compensate the shortage of useful data in the target project, in order to build a meaningful classification model. However, the distribution gap between software features extracted from the source and the target projects may be too large to make the mixed data useful for training. In this paper, we propose a cluster-based novel method FeSCH (Feature Selection Using Clusters of Hybrid-Data) to alleviate the distribution differences by feature selection. FeSCH includes two phases. The feature clustering phase clusters features using a density-based clustering method, and the feature selection phase selects features from each cluster using a ranking strategy. For CPDP, we design three different heuristic ranking strategies in the second phase. To investigate the prediction performance of FeSCH, we design experiments based on real-world software projects, and study the effects of design options in FeSCH (such as ranking strategy, feature selection ratio, and classifiers). The experimental results prove the effectiveness of FeSCH. Firstly, compared with the state-of-the-art baseline methods, FeSCH achieves better performance and its performance is less affected by the classifiers used. Secondly, FeSCH enhances the performance by effectively selecting features across feature categories, and provides guidelines for selecting useful features for defect prediction.

Keywords software defect prediction, cross-project defect prediction, feature selection, feature clustering, density-based clustering

1 Introduction

Software defects may be introduced during different stages of software development, resulting in defects hidden in software codes. The defects may come from misunderstanding of the requirements, uncontrollable development process, or the lack of developer experience. Hidden defects will produce unexpected results or failures after deployment. Software testing and code inspection are effective ways in detecting hidden defects. Since resources of software testing and code inspection are usually limited, methods are required to identify potential defective software modules which lead to efficient use of the resources.

Software defect prediction (SDP) is an active research problem in software engineering for identifying defect-prone software modules. SDP usually takes the following steps in estimating the degree of defect-proneness. Firstly, software modules are extracted from software historical repositories. Secondly, software metrics (i.e., features) are extracted from the modules and the corresponding development processes, along

Regular Paper
Special Section on Software Systems 2017
This work is supported in part by the National Natural Science Foundation of China under Grant Nos. 61373012, 91218302, 61321491 and 61202006, the Collaborative Innovation Center of Novel Software Technology and Industrialization, the Open Project of State Key Laboratory for Novel Software Technology at Nanjing University under Grant No. KFKT2016B18, and the National Basic Research 973 Program of China under Grant No. 2009CB320705.
*Corresponding Author

with labels identifying whether the modules are defect-prone. Thirdly, classification models are trained based on the labeled data, which are used later to predict defect-proneness of new software modules. With the aid of SDP, the limited testing resources can hopefully be allocated in a cost-effective manner.

The majority of existing SDP researches focus on within-project defect prediction (WPDP), which builds the prediction model and predicts defects within the same project. This hinders the application of SDP, because the target project may not have sufficient labeled modules for training any useful classification model. A feasible solution is to use data from other projects (called the source projects) to train classifiers used on the target project, which leads to the cross-project defect prediction (CPDP)[1-2]. To make the data from source projects useful, the problem is how to minimize the gap between different projects, normally represented by dissimilar feature distributions[3].

To alleviate the distribution differences between the source and the target project data, transfer learning methods have been studied for CPDP in recent years. Generally, knowledge from source projects can be transferred in two ways: instance transfer[4-5] and feature transfer[1,6]. Nam *et al.*[1] proposed the merits of feature transfer for CPDP. Their experiments proved that the distribution gap could be relieved by feature mapping and feature selection.

In this paper, we propose a novel feature selection method FeSCH for CPDP which has two phases. In the feature clustering phase, we cluster redundant features into multiple clusters. In the feature selection phase, we choose features, which have similar distribution between the source project and the target project, from each cluster to construct the target feature set. In particular, FeSCH firstly uses a density-based clustering method DPC (Density Peaks Clustering)[7] to divide the original feature set into multiple clusters. Then it uses a specific ranking strategy to select appropriate features from each cluster. For CPDP, we design three different heuristic strategies in the feature selection phase, which are LDF (local density of features), SFD (similarity of feature distributions), and FCR (feature-class relevance) respectively.

The main contributions of this paper can be highlighted as follows.

• We propose a cluster-based feature selection method FeSCH, which uses a density-based clustering method with three ranking strategies to select useful features for CPDP.

• We conduct empirical studies based on real-world software projects to demonstrate the effectiveness of FeSCH, which can provide useful guidelines for real-world applications.

The rest of the paper is organized as follows. Section 2 introduces the related work of our research. Section 3 describes the proposed method FeSCH. Section 4 presents the experimental setup. Section 5 analyzes the experimental results. Section 6 explains the possible threats to validity. Finally, Section 7 concludes the paper with future work.

2 Related Work

2.1 Cross-Project Defect Prediction

Since the target software projects usually lack the labeled modules for training meaningful defect predictors, historical projects are usually required to build the source data for training. This situation is called the cross-project defect prediction (CPDP)[1]. The characteristics of the target and source projects are usually different, which makes CPDP a challenging task. In early studies, researchers have done empirical studies under CPDP and their results were not positive. Briand *et al.*[8] used the predictor trained on the Xpose project to predict modules in the Jwriter project. They found the performance was much worse than that of within-project defect prediction (WPDP). Zimmermann *et al.*[9] conducted 622 cross-project defect predictions in which only 21 predictions achieved acceptable performance. He *et al.*[10] observed that only 0.32%∼4.67% of trained predictors under the context of CPDP can achieve satisfactory performances.

Recently, researchers have proposed different approaches to improve the defect prediction performance under CPDP context. Turhan *et al.*[4] proposed the nearest neighbour filter which selects 10 most similar source instances (i.e., modules) for each target instance. Peters *et al.*[5] improved the filter design by taking in extra information from source projects. Ma *et al.*[11] proposed an approach which assigns high weights to the source instances that were similar to the target instances. Ryu *et al.*[12] proposed a hybrid instance selection method HISNN which uses the local and global knowledge from the nearest neighbors. Herbold *et al.*[13] introduced the concept of local models to CPDP. They found that local models only make a minor difference in comparison with global models and transfer learning for CPDP. Nam *et al.*[1] proposed TCA+, which uses the data normalization solution to

minimize the difference between the source and the target projects. Wang *et al.*[14] leveraged a representation-learning algorithm (i.e., deep learning) to learn semantic representation of the modules from the projects. Chen *et al.*[15] proposed a novel algorithm to remove irrelevant instances from the source projects. Canfora *et al.*[16] used genetic algorithm to optimize two different objectives (i.e., high recall and low code inspection cost) under CPDP context. Panichella *et al.*[17] combined defect prediction results from different classifiers. Zhang *et al.*[18] considered ensemble learning to combine different classifiers. Zhang *et al.*[19] proposed context-aware rank transformations that map feature distributions between the source and the target projects. Xia *et al.*[20] proposed a two-layer framework, which combines the genetic algorithm and ensemble learning to capture common properties between the source and the target projects and combine merits of multiple classifiers[1,5,17]. Herbold[21] developed a tool, which includes many state-of-the-art techniques under CPDP context, in order to facilitate replication for cross-project defect prediction.

Some researchers considered the case where the source and the target projects have different feature sets. Nam and Kim[22] proposed the heterogeneous defect prediction method to handle the differences in feature sets. Jing *et al.*[23] solved the problem by defining unified feature space and used CCA (Canonical Correlation Analysis)-based transfer learning.

Some researchers also took the change-level defect prediction into consideration, which is also referred to as Just-In-Time (JIT) defect prediction. Kamei *et al.*[24] made an empirical study on eleven open source projects, and found that although JIT models rarely performed well under the CPDP context, their performance could be improved by feature selection.

Recently, researchers have adopted search-based technologies and unsupervised learning methods for CPDP. Hosseini and Turhan[25] proposed a search based instance selection method for CPDP. Nam and Kim[26] performed defect prediction on unlabeled data using cluster based auto-labeling. Zhang *et al.*[27] designed a connectivity-based unsupervised classifier using only the unlabeled target project data. Clustering methods are useful based on the fact that defective software modules usually have excessive feature values or distinctive value patterns.

In this paper, for CPDP, we assume the source and target projects have similar feature sets, and propose a clustering-based feature selection method to alleviate the difference of feature distributions between the source and the target software projects.

2.2 Feature Selection

There is the problem of the curse of dimensionality in defect prediction datasets. That is, the data built for SDP usually contain redundant or irrelevant features, which may lead to poor prediction performance, high model complexity, and extra training time. Feature selection is an effective way to alleviate the curse of dimensionality in software defect prediction, and can improve the performance of defect predictors[28]. Feature selection is a process of identifying and removing irrelevant and redundant features from the data so that only beneficial features are left for training. Existing feature selection methods can be roughly grouped into two categories: filter-based and wrapper-based. The filter-based methods evaluate and select the high relevant features, considering the correlation between the features and the class label. These approaches are independent of the classification models. The wrapper-based methods require feedbacks from the prediction results, which are used to improve iteratively the selected feature set. Gao *et al.*[28] applied feature selection for defect prediction on a large-scale legacy software system in telecommunications. Shivaji *et al.*[29] applied feature selection in change-based defect prediction. Xu *et al.*[30] used a double Scott-Knott test technique to compare the performances of 32 state-of-the-art feature selection methods. Their results suggested that the performance of the feature selection methods varies significantly on different datasets. Xu *et al.*[31] introduced principal component analysis (PCA) and consistent index to investigate the equivalence among different feature selection methods. Ghotra *et al.*[32] conducted large-scale empirical studies to analyze the impact of feature selection, and proved the merits of feature selection techniques.

Feature selection technologies have been studied under CPDP context. Amasaki *et al.*[6] devised methods to remove irrelevant features and unrelated instances from source project data according to the target project. Nam and Kim[22] proposed a method to handle the homogeneous cross-project defect learning, which combines feature mapping and feature selection.

Our previous work[33-35] proposed a cluster-based feature selection framework and a two-phase data preprocessing approach to improve the performance of defect predictors under the context of WPDP. In this paper, we extend our previous method considering the

requirements of CPDP. The main difference between our previous work and current work can be summarized as follows. 1) Our previous work mainly concerns on WPDP context, while our current work considers CPDP context. 2) In the feature selection phase, our previous work extended k-medoids method to perform feature clustering. This method needs to specify the number of clusters in advance. However, in current work, we use DPC to cluster features. This method is more flexible since it does not need to define the number of features in advance. 3) To effectively identify and remove redundant features and features with large distribution difference between the source projects and the target projects under CPDP context, we propose three new feature ranking strategies, i.e., LDF (local density of features), SFD (similarity of feature distributions), and FCR (feature-class relevance).

3 FeSCH

To alleviate the distribution gap between the source project and the target project, we propose a novel method FeSCH (Feature Selection Using Clusters of Hybrid-Data), which selects suitable features based on clustering across cross-project data. The framework of FeSCH is shown in Fig.1.

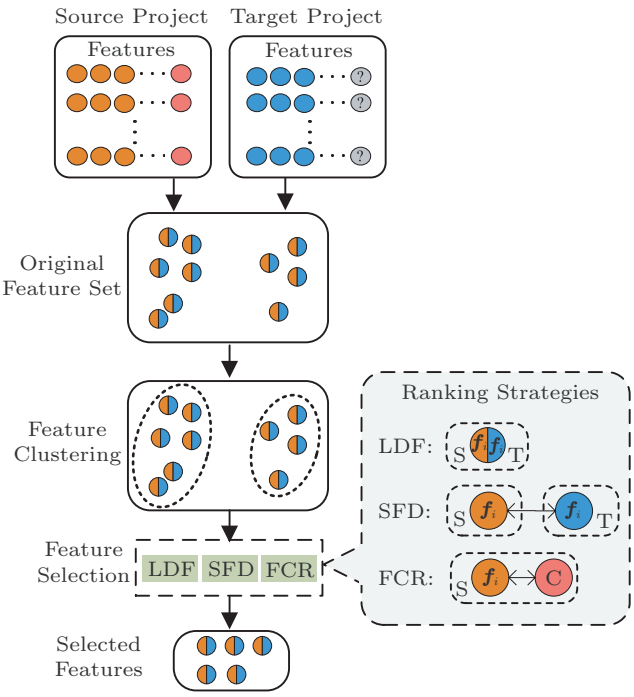


Fig.1. Framework of FeSCH. S: source project; T: target project; C: the class of the target project.

In detail, FeSCH contains two phases: the feature clustering phase and the feature selection phase. In the former phase, we use the density-based clustering method DPC[7] to divide the original feature set into multiple clusters according to the correlation among the features. Features are highly correlated with each other in the same cluster, while they are irrelevant in different clusters. In the latter phase, we rank the features in each cluster using a specific strategy to select the appropriate features from each cluster. In order to reflect the population of each cluster, we will select features from each cluster proportionally on account of the size of the cluster.

After using FeSCH, a classifier is trained on the cross-project data using only selected features as the defect predictor. In this paper, we choose the Logistic Regression as the classifier, which is commonly used and whose performance can be guaranteed[1,22]. In Section 5, we will prove that the performance of FeSCH is largely independent of the classifiers used.

3.1 Feature Clustering Phase

In this phase, we use DPC (density peaks clustering)[7] to cluster features using data from both the source and the target software projects. DPC can divide the features into non-predetermined number of clusters. A cluster center has two characteristics: high density and large distance. The high density requires that a cluster center should be surrounded by features with low density. The large distance requires that the center should have a large distance to features with great densities. Three steps are involved in the feature clustering phase using DPC.

In the first step, for each feature $\boldsymbol{f}_i$, its local density ρ_i and local distance δ_i are calculated. Let d_{ij} be the Euclidean distance of the features $\boldsymbol{f}_i$ and $\boldsymbol{f}_j$, and it can be computed as follows:

$$d_{ij} = \sqrt{\sum_{k=1}^{m}(x_{i_k} - x_{j_k})^2},$$

where m is the number of instances from the dataset, and x is the feature value in an instance. The local density ρ_i is the number of features within a predetermined distance:

$$\rho_i = \sum_j \chi(d_{ij} - d_c),$$

where $\chi(x) = 1$ if $x < 0$ and $\chi(x) = 0$ otherwise, and d_c is the cutoff distance which ensures each feature has at least 2% of the total features as its neighbors.

The local distance δ_i is the Euclidean distance from feature $\boldsymbol{f}_i$ to the nearest feature with greater local density:

$$\delta_i = \min_{j:\rho_j > \rho_i} (d_{ij}).$$

For the features with the greatest local density, their local distances are the greatest possible Euclidean distances:

$$\delta_i = \max_j(d_{ij}), \text{ if } \boldsymbol{f}_i \text{ has the maximum } \rho_i.$$

In the second step, we determine centers of the feature clusters using the calculated local densities and local distances. Firstly, for the feature $\boldsymbol{f}_i$, we calculate the centrality measure γ_i as follows:

$$\gamma_i = z(\rho_i) \times z(\delta_i),$$

where $z(\cdot)$ is the z-score normalization function. Secondly, we sort the features in descending order of γ_i, denoted as $\{\gamma\}_i^n$, where n is the total number of features. Based on $\{\gamma\}_i^n$, which conforms to the power law distribution, we can use its single inflection point as the cut point γ_c. The features with centrality measures greater than γ_c are used as the cluster centers.

In the last step, suppose there are k centers determined, each of the remaining features is assigned to the nearest center which has greater local density. By using DPC, the features are clustered in one round and k does not need to be pre-specified.

3.2 Feature Selection Phase

In this phase, three ranking strategies are designed to rank the features in each cluster. Since clusters have non-similar size, and the number of clusters is not predetermined, the number of selected features is proportional to the cluster size (e.g., more features will be selected from larger clusters). Such handling may introduce redundancy, but it keeps the number of selected features as promised. The three strategies are defined as follows.

3.2.1 LDF (*Local Density of Features*) *Strategy*

In this strategy, the features are selected using the local density measure. The local density means the number of neighbors around a feature, which may reflect the degree of popularity or representativeness of the feature.

3.2.2 SFD (*Similarity of Feature Distributions*) *Strategy*

In this strategy, the features are selected based on the level of distribution similarity between the source and the target project data. We use MIC (maximal information coefficient)[36] to measure the difference of two distributions. Given two vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$, which correspond to the same feature from the source project and the target project respectively, $MIC(\boldsymbol{X}, \boldsymbol{Y})$ can be computed as follows:

$$MIC(\boldsymbol{X}, \boldsymbol{Y}) = \max_{g_x \times g_y < n^{0.6}} \frac{\max\limits_{\boldsymbol{G} \in (g_x, g_y)} I(\boldsymbol{X}(\boldsymbol{G}), \boldsymbol{Y}(\boldsymbol{G}))}{\log \min(g_x, g_y)},$$

where $\boldsymbol{G}$ belongs to the grids generated by multiple $g_x \times g_y$ combinations, g_x and g_y are numbers of the equal partitions of vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ respectively, and n is the search range of $\boldsymbol{G}$[36]. For each grid $\boldsymbol{G}$, $\boldsymbol{X}(\boldsymbol{G})$ and $\boldsymbol{Y}(\boldsymbol{G})$ are vectors of the mean values of the partitions. We set the maximum possible $I(\boldsymbol{X}(\boldsymbol{G}), \boldsymbol{Y}(\boldsymbol{G}))$ as the MIC value. $I(\cdot)$ is the function of mutual information computed as follows:

$$I(\boldsymbol{X}, \boldsymbol{Y}) = H(\boldsymbol{X}) + H(\boldsymbol{Y}) - H(\boldsymbol{X}, \boldsymbol{Y}),$$

where $H(\cdot)$ is the entropy function, which is computed as follows:

$$H(\boldsymbol{X}) = -\sum_{x \in \boldsymbol{X}} p(x) \log_2 p(x). \quad (1)$$

3.2.3 FCR (*Feature-Class Relevance*) *Strategy*

This strategy has been used in our previous work FECAR[33]. The main assumption is that the features strongly related to the class labels are worth selecting. Under CPDP context, since the target data have no class labels, we use IG (information gain) to measure the feature-to-label relevance based on the source data. IG can be computed as follows:

$$IG(\boldsymbol{X}|\, \boldsymbol{Y}) = H(\boldsymbol{X}) + \sum_{y \in \boldsymbol{Y}} p(y) \sum_{x \in \boldsymbol{X}} p(x|y) \log_2 p(x|y),$$

where $\boldsymbol{X}$ is the feature vector from the source data, $\boldsymbol{Y}$ is the class label vector, x and y are the element values, and $p(\cdot)$ is the probability function. $H(\cdot)$ is the entropy function computed by (1).

3.2.4 *Comparison Among the Feature Ranking Strategies*

Comparing the three ranking strategies, LDF combines the source and the target data as a whole, and se-

lects the representative features. SFD makes a comparison between the source and the target data, and selects the features which have similar distributions between these two datasets. FCR only considers the source data, and selects features which have high relevance to the class label. In our empirical studies, we will further compare these ranking strategies. Although multiple features may be selected from a cluster, the latter two strategies select features from a different perspective, and hence can reduce the possible redundancy. During experiments, we use the three strategies separately, and compare their effects on the performance of the defect predictor trained for CPDP.

3.3 Time Complexity Analysis

Algorithm 1 summarizes our proposed method FeSCH under CPDP context. We assume the source and target projects share similar feature sets, which can be achieved by the repository mining procedures. The time complexity of FeSCH is $O(n'^2 \times (m + m'))$, where n' represents the number of selected features, and m and m' represent the number of instances in the source project and the target project respectively.

Algorithm 1. FeSCH

Input:
$\boldsymbol{S}^{m\times n}$: data from the source projects;
$\boldsymbol{T}^{m'\times n}$: data from the target project;
n_t: the specified number of selected features;
strat: the feature ranking strategy;
Output:
FS: the final set of selected features
`/* Feature Clustering Phase */`
1 Combine $\boldsymbol{S}^{m\times n}$ and $\boldsymbol{T}^{m'\times n}$ as the dataset $\boldsymbol{D}$
2 **for** each feature vector $\boldsymbol{f}_i$ in $\boldsymbol{D}$ **do**
3 Compute the density and distance measures ρ_i and δ_i
4 Compute the centrality measure $\gamma_i = \rho_i \times \delta_i$
5 **end**
6 Sort the set $\{\gamma\}_i^n$ in descending order
7 Compute the cut point γ_c based on the sorted $\{\gamma\}_i^n$
8 Label the features above γ_c as the cluster centers
9 Assign the rest features below γ_c to appropriate clusters
`/* Feature Selection Phase */`
10 **for** each cluster C **do**
11 Sort its features in descending order according to *strat*
12 **end**
13 $FS \leftarrow \emptyset$
14 **for** each cluster C **do**
15 Select top $\lceil \frac{|C|\times n_t}{n} \rceil$ features of C into FS
16 **end**
17 **return** FS

In Algorithm 1, lines 1∼9 describe the procedure of the feature clustering phase, and lines 10∼16 describe the procedure of the feature selection phase. In line 15, features are selected from each cluster proportionally on account of the cluster size. For example, we assume the source project has n features and we want to select n_t features from the original features by using our method. Then for a cluster C, which has $|C|$ features in this cluster, we will choose top $\left\lceil \frac{|C|\times n_t}{n} \right\rceil$ features from this cluster.

4 Experimental Setup

In this section, we firstly present the research questions for the empirical study. Secondly, we describe the experimental design. Thirdly, we present the datasets which are widely used under CPDP context, including the feature categories in each dataset. Lastly, we introduce the performance measures used for comparison.

4.1 Research Questions

To verify the effectiveness of the proposed FeSCH, we design experiments to investigate the following research questions.

*RQ*1. Can FeSCH further improve the performance under CPDP context when compared with the baseline methods?

*RQ*2. How do the design options (i.e., the ranking strategy, the feature selection ratio, and the classifier) under FeSCH affect the performance of CPDP?

*RQ*3. Can FeSCH improve the performance of CPDP when compared with feature selection methods only based on feature category?

4.2 Experimental Design

To evaluate the performance of FeSCH under CPDP context, we use three baseline methods: WPDP (within-project defect prediction without feature selection), ALL (all features under CPDP context), and TCA+[1].

For WPDP, we train and test the defect predictor on the target project only, and use 2-fold cross validation (CV). We do not use 10-fold CV since, firstly, 2-fold CV is a common setting in CPDP[1,22], and secondly, 90% labeled instances available in the target project make an unfair comparison under CPDP context. To overcome possible bias in the random selection processes, we repeat the random CV split 100 times and report the

average prediction results. For ALL, we train the defect predictor using all features from the source project, and use the trained predictor to make prediction on the target project. For TCA+, we directly use the results reported by Nam *et al.*[1]

We will examine the influence of different classifiers on the performance of FeSCH. Three classifiers are considered, including the probability-based classifier Naive Bayes, decision-tree based classifier Random Forest, and function-based classifier Logistic Regression. To facilitate comparison with the baseline methods, Logistic Regression is used as the default classifier. Logistic Regression is commonly used in previous research[1,22]. During experiments, we use the implementation provided by LibLinear[37] and consider the option (i.e., −S 0 −B 1) used in TCA+[1]. To calculate the relevance formulas, we use the MINE① (Maximal Information-based Nonparametric Exploration) toolkits[36] with default setting.

4.3 Datasets

During experiments, we use two classical datasets for CPDP[1]: ReLink and AEEEM. Table 1 lists the details of these datasets.

The ReLink dataset was collected by Wu *et al.*[38] They used the Understand toolkit② to analyze and extract the value of software metrics from three projects (i.e., Apache, Safe and ZXing). ReLink has 26 features, which are categorized into the complexity metrics and the count metrics. These metrics are designed based on the code complexity (such as LOC, cycloramic complexity, and the number of classes), and abstract syntax trees (such as the number of blocks, the number of statements, and method references).

Table 1. Details of Datasets

Dataset	Project	Level	Number of Features	Number of Instances	Number of Faulty Instances (Percent)
ReLink	Apache	File	26	194	98 (50.52%)
	Safe			56	22 (39.29%)
	ZXing			399	118 (29.57%)
AEEEM	EQ	Class	61	324	129 (39.80%)
	JDT			997	206 (20.70%)
	LC			691	64 (9.30%)
	ML			1 862	245 (13.20%)
	PDE			1 497	209 (14.00%)

The AEEEM dataset contains five projects: EQ, JDT, LC, ML, and PDE, which were collected by D'Ambros *et al.*[39] AEEEM has 61 metrics, which are categorized into code metrics, process metrics, and change-based metrics. Artificial metrics are computed using facilities such as entropy.

4.4 Feature Categories

The features of ReLink and AEEEM are grouped into different categories. In ReLink, the extracted features are grouped into two categories according to the definition in Understand③. There are 12 complexity metrics and 15 count metrics. There is such a fact that the sum (i.e., 27) of the number of complexity metrics (i.e., 12) and the number of count metrics (i.e., 15) is greater than the total number of features (i.e., 26). The reasoning behind the fact is that the feature in ReLink named "RatioCommentToCode" belongs to two categories simultaneously. In AEEEM[39], the 61 features are grouped into five categories: 5 previous defect metrics, 5 entropy of changes metrics, 17 source code metrics, 17 entropy of source code metrics, and 17 churn of source code metrics. Table 2 gives the details of these feature categories.

4.5 Performance Measures

According to the ground truths and the predicted class labels, we can compute the number of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) respectively. Then, we can

①http://www.exploredata.net/Downloads/MINE-Application/, Sept. 2017.

②https://scitools.com/, Sept. 2017.

③https://scitools.com/support/metrics_list, Sept. 2017.

Table 2. Feature Categories in ReLink and AEEEM

Dataset	Category	Description	Number of Features
ReLink	ComplexityMetric (CPM)	Complexity measures of the source code, e.g., McCabe Cyclomatic measure	12
	CountMetric (CTM)	Quantitative counting of the source code, e.g., the number of all lines	15
AEEEM	SourceCodeMetric (SCM)	Metrics solely computed from source code	17
	ChurnOfSourceCodeMetric (COSCM)	Artificial metrics computed from SCM based on code churn	17
	EntropyOfSourceCodeMetric (EOSCM)	Artificial metrics computed from SCM based on entropy	17
	PreviousDefectsMetric (PDM)	Metrics computed from known defects and the revisions	5
	EntropyOfChangesMetric (EOCM)	Artificial metrics computed from changes based on entropy	5

calculate the precision measure p and recall measure r as follows:

$$p = \frac{TP}{TP + FP},\ r = \frac{TP}{TP + FN}.$$

There always exists a trade-off between the precision measure and the recall measure. To handle the dilemma, the $F1$-measure, which will be used in our experiments, is defined as follows:

$$F1 = \frac{2 \times p \times r}{p + r}.$$

To extensively compare the performance of FeSCH and the baseline methods, we also use AUC measure. AUC measures the area under the receiver operating characteristic (ROC) curve, which is a 2D illustration of true positive rate on the y-axis versus false positive rate on the x-axis. ROC curve is obtained by varying the classification threshold over all possible values, separating clean and buggy predictions. A well preformed predictor provides an AUC value close to 1. The ROC analysis is robust in case of imbalanced class distributions and asymmetric misclassification costs, which is suitable for CPDP.

To check the significance of performance comparison, we conduct the Wilconxon signed-rank test, which is a non-parametric statistical hypothesis test on the performance measures. For all the tests, the null hypotheses are that there is no difference between the trained predictors, and the significance level α is set to 0.05. If p-value is smaller than 0.05, we reject the null hypotheses; otherwise we accept the null hypotheses.

To perform a further comparison of pairwise methods, we also use "Win/Draw/Loss" analysis. The "Win/Draw/Loss" record of "method 1 vs method 2" presents three values on a given measure. These three values are the number of datasets for which method 1 performs better than, equal to, and worse than method 2 respectively.

5 Experimental Results

5.1 Result Analysis for RQ1

For RQ1, we perform FeSCH on ReLink and AEEEM datasets respectively, using the Logistic Regression as the classifier, and compare its performance with that of three baseline methods. Table 3 and Table 4 present the final results.

In both tables, the first column presents specific cases of CPDP. For example, in Table 3, the case "Safe⇒Apache" means the project "Safe" is used as

Table 3. Comparison of $F1$-Measure Among Baseline Methods, FeSCH, and Feature Categories on ReLink

Source⇒Project	Baseline Method			FeSCH			Category	
	WPDP	ALL	TCA+	LDF	SFD	FCR	CPM	CTM
Safe⇒Apache	0.68	0.52	0.64	0.65	**0.65**	0.62	0.49	0.54
ZXing⇒Apache	0.68	0.68	**0.72**	0.69	0.61	0.67	0.59	0.57
Apache⇒Safe	0.60	0.56	**0.72**	0.22	0.62	0.52	0.71	0.36
ZXing⇒Safe	0.60	0.59	0.64	0.64	**0.71**	0.78	0.60	0.62
Apache⇒ZXing	0.31	0.46	0.49	0.13	**0.67**	0.62	0.56	0.33
Safe⇒ZXing	0.31	0.10	0.43	0.62	**0.62**	0.65	0.58	0.60
Average	0.53	0.49	0.61	0.49	**0.65**	0.64	0.59	0.50

Table 4. Comparison of $F1$-Measure Among Baseline Methods, FeSCH, and Feature Categories on AEEEM

Source⇒Project	Baseline Method			FeSCH			Category				
	WPDP	ALL	TCA+	LDF	SFD	FCR	SCM	COSCM	EOSCM	PDM	EOCM
JDT⇒EQ	0.58	0.30	0.60	0.41	**0.66**	0.52	0.40	0.29	0.27	0.41	0.27
LC⇒EQ	0.58	0.51	**0.62**	0.38	0.50	0.55	0.42	0.33	0.20	0.33	0.23
ML⇒EQ	0.58	0.24	**0.56**	0.37	0.46	0.53	0.45	0.30	0.18	0.43	0.26
PDE⇒EQ	0.58	0.43	**0.60**	0.41	0.55	0.49	0.42	0.31	0.20	0.42	0.31
EQ⇒JDT	0.53	0.39	**0.54**	0.22	0.32	0.33	0.28	0.19	0.10	0.12	0.06
LC⇒JDT	0.53	0.49	0.56	0.67	**0.62**	0.67	0.55	0.45	0.49	0.47	0.45
ML⇒JDT	0.53	0.42	0.43	0.63	**0.62**	0.61	0.39	0.49	0.49	0.37	0.44
PDE⇒JDT	0.53	0.47	0.48	0.60	**0.61**	0.59	0.55	0.50	0.49	0.43	0.44
EQ⇒LC	0.31	0.26	0.27	0.34	**0.45**	0.46	0.49	0.32	0.11	0.01	0.08
JDT⇒LC	0.31	0.26	0.31	0.51	**0.53**	0.53	0.55	0.49	0.47	0.26	0.14
ML⇒LC	0.31	0.10	0.25	0.47	**0.49**	0.48	0.57	0.51	0.55	0.27	0.12
PDE⇒LC	0.31	0.33	0.33	0.47	**0.52**	0.45	0.57	0.52	0.55	0.45	0.53
EQ⇒ML	0.26	0.19	0.23	0.33	**0.48**	0.17	0.37	0.35	0.14	0.01	0.01
JDT⇒ML	0.26	0.27	0.36	0.39	**0.49**	0.49	0.42	0.52	0.43	0.54	0.49
LC⇒ML	0.26	0.20	0.29	0.64	**0.43**	0.45	0.42	0.52	0.54	0.60	0.32
PDE⇒ML	0.26	0.28	0.29	0.75	**0.42**	0.80	0.43	0.55	0.54	0.49	0.12
EQ⇒PDE	0.33	0.36	0.33	0.12	**0.48**	0.41	0.46	0.21	0.10	0.26	0.08
JDT⇒PDE	0.33	0.28	0.38	0.51	**0.51**	0.51	0.49	0.48	0.34	0.31	0.48
LC⇒PDE	0.33	0.31	0.37	0.52	**0.49**	0.53	0.48	0.47	0.61	0.53	0.58
ML⇒PDE	0.33	0.27	0.37	0.50	**0.47**	0.43	0.49	0.51	0.60	0.45	0.34
Average	0.40	0.32	0.41	0.46	**0.51**	0.50	0.46	0.42	0.37	0.36	0.29

the source project, and the project "Apache" is used as the target project. The "baseline methods" column lists the results of three baseline methods (i.e., WPDP, ALL, and TCA+) and the "FeSCH" column lists the results using one of the three strategies. The last row provides the average performance in each table. The SFD ranking strategy is used as the adopted ranking strategy in our FeSCH due to its stable performance and the further investigation of the impact of different ranking strategies will be given in the next subsection. The feature selection ratio is set to 30%. For each case (row), the best result of FeSCH and the baseline methods is underlined, and the best result of CPDP methods (i.e., ALL, TCA+, and FeSCH) is emphasized in bold.

From the last row of the two tables, we can find that FeSCH can achieve the best performance among the three methods on both ReLink and AEEEM datasets. Especially, FeSCH has dominant performance on AEEEM. On both datasets, the performance of both TCA+ and FeSCH is nearly similar to that of WPDP with 50% labeled instances, which proves that the methods are promising for CPDP.

Compared with WPDP, FeSCH performs better in most cases. For example, in the case "ZXing ⇒ Safe" on ReLink, FeSCH has the best performance of 0.71, much greater than 0.60 of WPDP. In the case "LC ⇒ JDT" on AEEEM, FeSCH achieves the best performance of 0.62, greater than 0.53 of WPDP. The evidence strongly suggests that FeSCH makes an effective use of the source projects.

Compared with ALL, FeSCH performs better in nearly all the cases on both ReLink and AEEEM. The results prove that feature selection is essential for CPDP, conforming to the findings of Nam *et al.*[1] The possible reasons are that irrelevant and redundant features will cause serious performance degradation for software defect prediction, especially under CPDP context.

Compared with TCA+, FeSCH still performs better in the majority of cases on both ReLink and AEEEM. The results suggest that feature selection is promising under CPDP context, which can decrease the costs required in measuring software modules. However, FeSCH performs worse in several cases, such as "Apache", "Safe" of ReLink and "EQ" on AEEEM. The reason may be that these datasets contain less than 350 instances and may not provide enough information for feature selection. In these cases, the feature space projection used in TCA+ is more helpful.

The reasons why FeSCH performs better than

TCA+ can be summarized as follows. 1) Most classifiers are designed under the assumption that training and test datasets are represented in the same feature space and drawn from the same data distribution. Therefore, the difference in feature distribution between the source project and the target project may cause the poor performance under the CPDP context and even make the classifier invalidate. TCA+ aims to figure out a latent feature space for the data between the source and the target project by normalizing feature data. Then mapping the data of the source project and the target project into such latent space may cause the loss of original information. FeSCH can save the information loss by directly picking up features from original data according to similarity distribution of features. 2) Redundant features exist not only in the source project but also in the target project, which will not only increase the time complexity of model training, but also decrease the performance of the classifier. TCA+ does not consider the influence of redundancy, while the proposed FeSCH can effectively remove redundant features through feature clustering so that the performance is further improved.

Table 5 summarizes the results of "win/draw/loss" between FeSCH using SFD and the three baseline methods. The first two columns represent the datasets and target projects respectively. The rest columns represent the three baseline methods to be compared. Each row refers to one project in the particular dataset taken as the target project, and the rest projects in the same dataset served as the source project.

Table 5. Win/Draw/Loss of FeSCH (SFD) Compared with the Baselines on Both Datasets

Dataset	Target	Against (Win/Draw/Loss)		
		WPDP	ALL	TCA+
ReLink	Apache	0/0/2	1/0/1	1/0/1
	Safe	2/0/0	2/0/0	1/0/1
	ZXing	2/0/0	2/0/0	2/0/0
	Total	4/0/2	5/0/1	4/0/2
AEEEM	EQ	1/0/3	3/0/1	1/0/3
	JDT	3/0/1	3/0/1	3/0/1
	LC	4/0/0	4/0/0	4/0/0
	ML	4/0/0	4/0/0	4/0/0
	PDE	4/0/0	4/0/0	4/0/0
	Total	16/0/4	18/0/2	16/0/4

Taking the first row as an example, the ReLink has three projects (i.e., Apache, Safe and ZXing). When the "Apache" in ReLink was taken as the target project, the rest projects in ReLink (i.e., Safe and ZXing) were served as the source project respectively. Compared with the baseline methods (i.e., WPDP, ALL and TCA+), when the result of FeSCH is larger than the results of baselines, then FeSCH wins; when the result of FeSCH is smaller than the results of baselines, then FeSCH losses; otherwise they draw.

Relink has three projects, which leads to a total of six combinations. In particular, for each target project, it has two cases of CPDP. From Table 5, FeSCH has at least 4/6 wins on ReLink, and 16/20 wins on AEEEM, which demonstrates its effectiveness.

To check whether the performance differences among WPDP, ALL, TCA+ and FeSCH (SFD) are significant, we conduct the Wilconxon signed-rank test. Table 6 shows the results in detail. Combining all the cases of the two datasets, the p-values of "FeSCH vs WPDP", "FeSCH vs ALL", and "FeSCH vs TCA+" are all less than 0.05, which indicates that FeSCH is statistically better than the baseline methods.

Table 6. p-Value of the Wilconxon Signed-Rank Test Among Baseline Methods and FeSCH (SFD)

Dataset	FeSCH vs WPDP	FeSCH vs ALL	FeSCH vs TCA+
ReLink	1.24e-01	3.74e-02	2.32e-01
AEEEM	1.80e-03	7.00e-05	2.56e-03
Both datasets	5.03e-04	1.05e-05	1.75e-03

Summary. FeSCH can outperform the three baseline methods in most cases under the CPDP context and has improved performance compared with benchmark methods on average on these two datasets (25% improvement compared with WPDP, 46% improvement compared with ALL, and 15% improvement compared with TCA+). However, when the target project does not have enough data (e.g., less than 350 instances), TCA+ proves to be a better choice than FeSCH.

5.2 Result Analysis for RQ2

For RQ2, we study how the design options of FeSCH affect its performance under CPDP context and thus provide a guideline for using FeSCH. Here we focus on three design options: the ranking strategy, the feature selection ratio, and the trained classifier.

5.2.1 Effects of Ranking Strategies

In FeSCH, three ranking strategies, LDF, SFD and FCR, are designed for selecting representative features

from each cluster. The "FeSCH" column in Table 3 and Table 4 shows the results of FeSCH when using different ranking strategies on both datasets. Based on the average performance, FeSCH using either of the three ranking strategies outperforms WPDP on AEEEM, while FeSCH using SFD or FCR outperforms WPDP on ReLink. We also summarize the "win/draw/loss" information between WPDP and those different ranking strategies in Table 7. According to "win/draw/loss", the superiority of FeSCH becomes evident. All the three ranking strategies win in the majority of cases. On ReLink, FeSCH has at least 3/6 wins, while on AEEEM dataset, FeSCH has at least 14/20 wins. However, we observe that FeSCH performs poor when the project EQ in AEEEM is set as the target project. This implies that "EQ" is distinct in AEEEM, and further improvements are required for FeSCH.

Table 7. Win/Draw/Loss of WPDP Against the Three Ranking Strategies on ReLink and AEEEM

Dataset	Target	Against (Win/Draw/Loss)		
		LDF	SFD	FCR
ReLink	Apache	1/0/1	2/0/0	2/0/0
	Safe	1/0/1	0/0/2	1/0/1
	ZXing	1/0/1	0/0/2	0/0/2
	Total	3/0/3	2/0/4	3/0/3
AEEEM	EQ	4/0/0	3/0/1	4/0/0
	JDT	1/0/3	1/0/3	1/0/3
	LC	0/0/4	0/0/4	0/0/4
	ML	0/0/4	0/0/4	1/0/3
	PDE	1/0/3	0/0/4	0/0/4
	Total	6/0/14	4/0/16	6/0/14

Among the ranking strategies, SFD achieves the best average performance (i.e., 0.65 on ReLink and 0.51 on AEEEM), while LDF performs the worst (i.e., 0.49 on ReLink and 0.46 on AEEEM). No ranking strategy can win the other strategies in all the CPDP cases. For example, LDF still performs the best in one of the six cases on ReLink, and five of the 20 cases on AEEEM. On the other hand, SFD only performs the best in two cases on ReLink and nine cases on AEEEM. LDF performs the worst when the project "EQ" is set as the target project, which suggests that LDF is sensible to the difference between the source and the target projects. Considering the definitions of these ranking strategies, SFD selects features based on the similarity of feature distributions between the target and the source projects, and FCR considers the strong relevance between features and class labels on the source projects, while LDF calculates the Euclidean distances between features taking the source and the target projects as a whole. The results suggest that both distribution similarity and class relevancy are suitable for selecting features under CPDP, while directly combining the data of the source and the target projects may not work well.

We conduct the Wilconxon signed-rank test to check if the performance of the three ranking strategies is significantly different. Table 8 shows the results in detail. From this table, it can be seen that the performance differences among the three strategies are not always significant, which indicates that all the strategies are useful and have their own advantages on different datasets.

Table 8. p-Value of the Wilconxon Signed-Rank Test Among Three Ranking Strategies

Dataset	LDF vs SFD	LDF vs FCR	SFD vs FCR
ReLink	1.47e-01	7.11e-02	5.00e-01
AEEEM	4.83e-02	6.29e-02	3.34e-01
Both datasets	1.70e-02	1.86e-02	3.61e-01

Summary. All the three ranking strategies are effective and useful for FeSCH. The strategy SFD, which is to increase the similarity of feature distributions between the source and the target projects, performs better in our experiments. The strategy LDF, although performing the worst, is still promising in certain cases.

5.2.2 *Effects of Feature Selection Ratio*

To study the effects of feature selection ratio on FeSCH, we conduct experiments on both datasets varying the feature selection ratio from 5% to 100% with a step of 5%. The ranking strategy used is SFD. Fig.2 and Fig.3 show the results on both ReLink and AEEEM respectively.

In both figures, the x-axis indicates the feature selection ratio, and the y-axis indicates the corresponding $F1$-measure. The trend lines represent the mean $F1$-measure on different target projects, where the variances are caused by using different source projects.

As shown in Fig.2, selecting 30% of the original features can achieve good-enough performance on ReLink, which is better than using all the features. For example, when setting the project "Safe" as the target project, FeSCH can achieve the best average performance (0.67), which is better than using all the features (0.60). In addition, we find that selecting 20%∼40%

of the original features will achieve good performance. This finding is similar to the finding in [33].

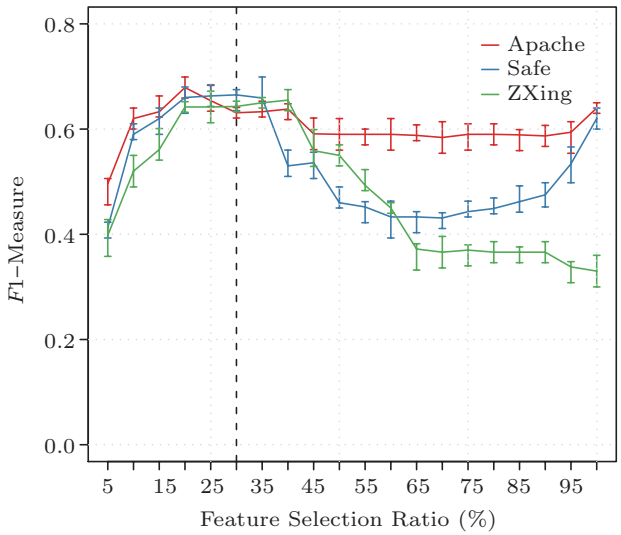


Fig.2. Average performance of FeSCH by varying feature selection ratio on ReLink.

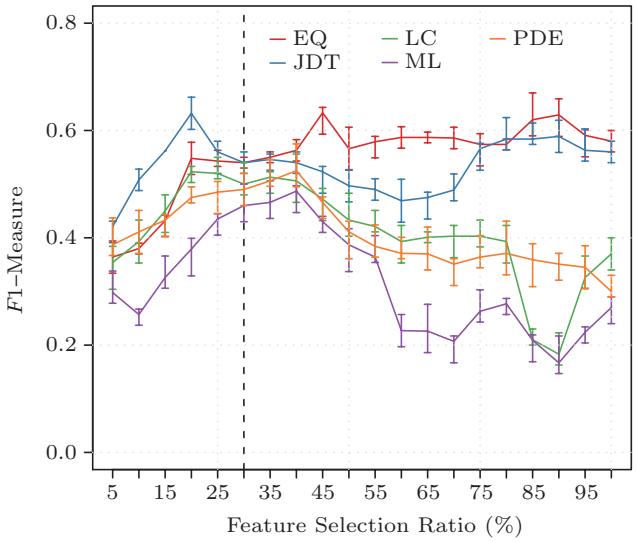


Fig.3. Average performance of FeSCH by varying feature selection ratio on AEEEM.

As shown in Fig.3, the optimal feature selection ratios on AEEEM are nearly the same as those on ReLink. When "LC", "PDE", or "ML" is set as the target project respectively, selecting 20%∼40% features is preferred. When "JDT" is set as the target project, the optimal selection ratio is about 20%. On the other hand, when "EQ" is set as the target project, the result is exceptional: the optimal ratio has a long range from 20% to 90%.

Summary. For FeSCH, the feature selection ratio can be set within 20%∼40%, and 30% can be a promising start point, at least on ReLink and AEEEM. Other CPDP cases are still required to provide sound evidence.

5.2.3 *Effects on Different Classifiers*

To check the performance of FeSCH when training different classifiers, we make a comparison among three commonly used classifiers: LR (Logistic Regression), NB (Naive Bayes), and RF (Random Forest). The ranking strategy used is SFD, and the feature selection ratio is 30%. Fig.4 and Fig.5 show the average performance on different target projects.

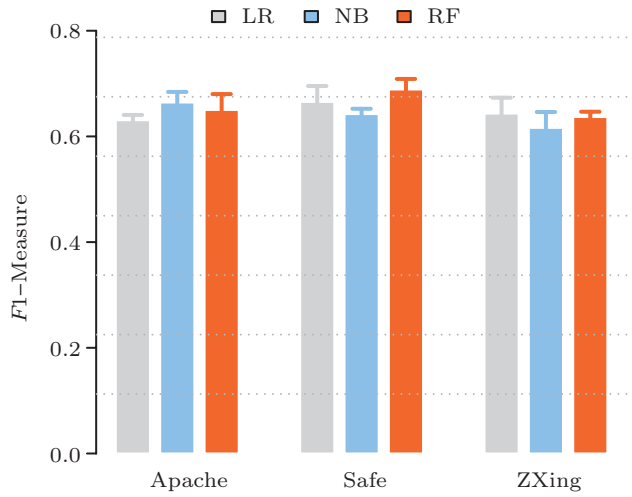


Fig.4. Comparison of the classifiers after using FeSCH on ReLink.

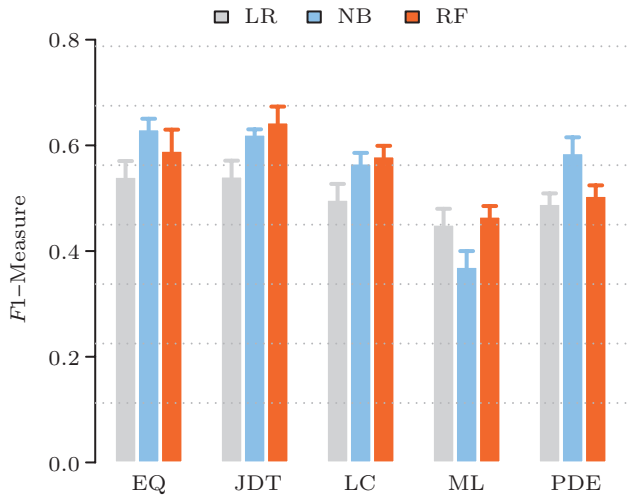


Fig.5. Comparison of the classifiers after using FeSCH on AEEEM.

In both figures, the x-axis indicates the target projects, and the y-axis indicates the corresponding $F1$-measure. Each bar represents the mean $F1$-measure on a specific target project resulted from one of the three classifiers.

As shown in Fig.4, on ReLink, we can hardly detect significant differences among the three classifiers. For example, the classifier NB performs the best when Apache is set as the target project. But NB performs the worst when either of the other two projects is set as the target project.

As shown in Fig.5, on AEEEM, the $F1$-measure varies a lot among the five target projects. For example, NB performs the best on "EQ" and "PDE", while RF performs the best on "JDT", "LC" and "ML". LR looks inferior, but the decrease in $F1$-measure is usually less than 0.07. The above suggests that the performance of FeSCH does not rely on the classifiers used, at least on the two datasets.

Summary. By applying FeSCH, the classifiers may not have discriminative effects on the prediction performance under the CPDP context. Researchers should pay attention to building high-quality source data for training useful defect predictors for the target project. In the future, we should consider more projects and investigate more classifiers to consider the generalization of our empirical results.

5.3 Result Analysis for RQ3

In order to investigate the effects of singular category of software metrics (features) under CPDP, we conduct experiments using one feature category at a time, and compare the results of ALL and FeSCH (SFD) respectively. The "category" columns of Table 3 and Table 4 list the results on the two datasets.

Compared with ALL, it is clear that using all features may not guarantee the best performance, and the prediction model constructed on certain feature category can obtain better performance. For example, on ReLink, the category CPM has at least 4/6 wins against ALL, and the average performance is much better than that of ALL. On AEEEM, both SCM and COSCM have won against ALL in most cases, and the average performance is better. On both datasets, there are feature categories that significantly outperform the others under the CPDP context, which may explain why ALL cannot obtain good performance by simply combining all the feature categories.

Compared with FeSCH (SFD), it is clear that FeSCH can outperform any singular feature category on the two datasets. Table 9 and Table 10 present "win/draw/loss" of FeSCH (using SFD strategy and 30% of original features) against various feature categories on both datasets. In both tables, each row refers to one project taken as the target project, and one of the rest projects served as the source project. From these tables, we can see that FeSCH has at least 5/6 wins against the best feature category on ReLink, and 14/20 wins against the best category on AEEEM, which demonstrate the effectiveness of FeSCH. The reason is that FeSCH can make good use of features from different categories to obtain better performance.

Table 9. Win/Draw/Loss of FeSCH (SFD) Against Different Feature Categories on ReLink

Target	Against (Win/Draw/Loss)	
	CPM	CTM
Apache	2/0/0	2/0/0
Safe	1/0/1	2/0/0
ZXing	2/0/0	2/0/0
Total	5/0/1	6/0/0

Table 10. Win/Draw/Loss of FeSCH (SFD) Against Different Feature Categories on AEEEM

Target	Against (Win/Draw/Loss)				
	SCM	COSCM	EOSCM	PDM	EOCM
EQ	4/0/0	4/0/0	4/0/0	4/0/0	4/0/0
JDT	4/0/0	4/0/0	4/0/0	4/0/0	4/0/0
LC	0/0/4	2/1/1	2/0/2	4/0/0	3/0/1
ML	3/0/1	1/0/3	2/0/2	1/0/3	3/1/0
PDE	3/0/1	3/0/1	2/0/2	3/0/1	3/0/1
Total	14/0/6	14/1/5	14/0/6	16/0/4	17/1/2

Moreover, it is useful to investigate which features are more likely to be selected by FeSCH, and hence valuable for CPDP. Table 11 and Table 12 list the top 10 features selected by FeSCH from ReLink and AEEEM respectively. In both tables, the first three columns represent the name, description and category of the selected features respectively, and the last column lists the selection frequency by FeSCH. The frequency of selection is the proportion of appearance in the selected feature sets of all the cases, which is used to order the features.

On ReLink, AvgCyclomaticModified and CountLineCodeExec are always selected by FeSCH. In general, features from the CTM category are mostly selected corresponding to the good performance of CTM. On AEEEM, among the top selected features, the source code based features (SCM, COSCM, and EOSCM) are

Table 11. Top 10 Features Selected by FeSCH on ReLink

Feature Name	Description	Category	Selection Frequency
AvgCyclomaticModified	Average modified cyclomatic complexity for all nested functions or methods	CPM	1.00
CountLineCodeExe	Number of lines containing executable source code	CTM	1.00
CountLineCode	Number of lines containing source code	CTM	0.78
CountLine	Number of all lines	CTM	0.67
CountSemicolon	Number of semicolons	CTM	0.67
CountStmt	Number of statements	CTM	0.67
CountLineBlank	Number of blank lines	CTM	0.56
CountLineCodeDecl	Number of lines containing declarative source code	CTM	0.56
MaxCyclomaticStrict	Maximum strict cyclomatic complexity of nested functions or methods	CPM	0.56
CountStmtDecl	Number of declarative statements	CTM	0.45

Table 12. Top 10 Features Selected by FeSCH on AEEEM

Feature Name	Description	Category	Selection Frequency
CvsEntropy	Entropy of source code	EOCM	0.92
CvsLogEntropy	Logarithmic variants of entropy of source code	EOCM	0.92
WCHU_numberOfLinesOfCode	Weighted churn of the number of lines of code	COSCM	0.92
CvsWEntropy	Weighted variants of entropy of source code	EOCM	0.87
ck_oo_numberOfLinesOfCode	Number of line code	SCM	0.85
ck_oo_rfc	Response for class	SCM	0.82
LDHH_numberOfAttributes	Linearly decayed entropy of the number of attributes	EOSCM	0.82
LDHH_cbo	Linearly decayed entropy of the coupling between objects	EOSCM	0.82
LDHH_numberOfLinesOfCode	Linearly decayed entropy of the number of lines of code	EOSCM	0.75
CvsLinEntropy	Linear variants of entropy of source code	EOCM	0.75

mostly selected. However, according to selection ratio, the entropy of changes-based features (EOCM) is dominant, since three out of five EOCM features appear in the top list. Entropy metric, digging the difference between successive software versions, is the most valuable for CPDP.

Summary. Different feature categories may present different performances under the CPDP context. Simply combining the feature categories may not ensure the best performance. The proposed method FeSCH can make good use of the features from multiple categories leading to a well-trained defect predictor for CPDP.

5.4 Discussions

We have performed an in-depth comparison between our proposed method FeSCH and a state-of-the-art method TCA+[1], which is based on feature mapping. To further investigate the effectiveness of FeSCH, we consider other types of CPDP methods for comparison: Peters[5], Burak[4] and HISNN[12]. Peters and Burak adapt the source project data by instance selection using the k nearest neighbors, and use the Euclidean distances to measure the differences among source and target instances. HISNN selects instances by clustering, and uses the Hamming distance to measure the difference.

Fig.6 and Fig.7 depict the results in box-plots, where the measures are taken from all cases on each dataset. In both figures, we group the box-plots by the four measures (i.e., precision, recall, $F1$-measure, and AUC), for comparison. The x-axis indicates the five methods arranged in four groups, while the y-axis indicates the corresponding scores of precision, recall, $F1$-measure or AUC. Each box-plot presents the mean, median, maximum, minimum, and quartiles of the specific measure. FeSCH uses the same settings as in RQ1. Logistic Regression is the classifier trained after all the methods.

Firstly, we illustrate the performance comparison on ReLink according to Fig.6. 1) Considering precision, the range between the quartiles of FeSCH is the smallest (i.e., 0.057) among all the methods (i.e., Peters is 0.225, Burak is 0.186, HISNN is 0.086 and TCA+ is 0.217). The mean score of FeSCH (i.e., 0.640) is also the best. 2) Considering recall, the mean score (i.e., 0.650) of FeSCH is again the best, and the quartile range (i.e., 0.023) is also the smallest among the

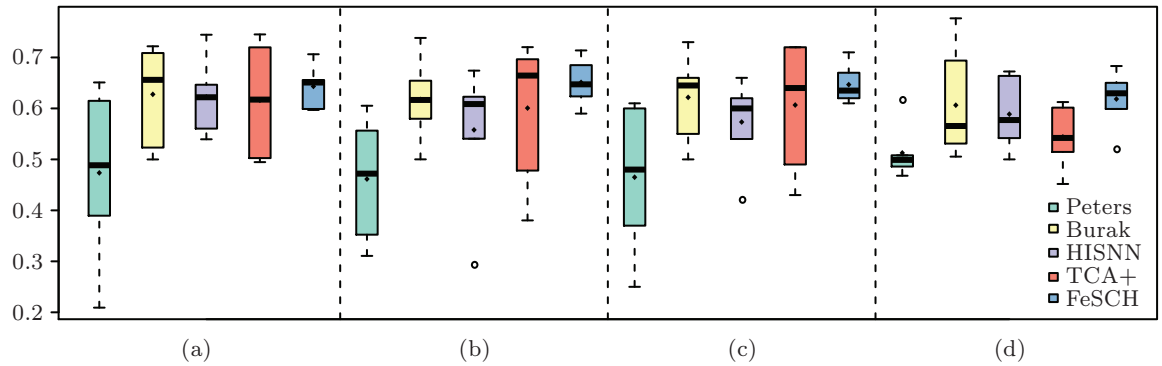


Fig.6. Comparison of FeSCH and four baseline methods using different measures on ReLink. (a) Precision. (b) Recall. (c) $F1$-measure. (d) AUC.

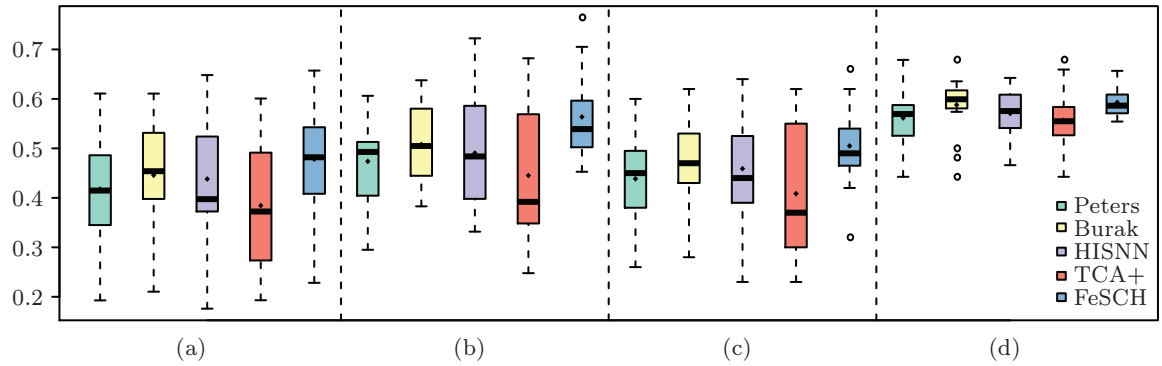


Fig.7. Comparison of FeSCH and four baseline methods using different measures on AEEEM. (a) Precision. (b) Recall. (c) $F1$-measure. (d) AUC.

methods. 3) Considering $F1$-measure, the mean score of FeSCH (i.e., 0.650) is the best because of the high precision and recall scores, and the quartile range (i.e., 0.050) remains the smallest among all the methods. 4) Considering AUC, FeSCH can also achieve relatively good results. The mean score (i.e., 0.620) is the best, while the quartile range is smaller than Burak, HISNN, and TCA+. Peters has the smallest quartile range, but the mean score (i.e., 0.510) is the worst.

Secondly, we illustrate the performance comparison on AEEEM according to Fig.7. 1) Considering both precision and recall, FeSCH achieves the best mean scores (i.e., 0.480 and 0.560), and its quartile ranges are also the smallest among all the methods. 2) Considering $F1$-measure, the mean score of FeSCH (i.e., 0.510) is the best, and the quartile range (i.e., 0.075) is the smallest among the methods. 3) Considering AUC, FeSCH and Burak have the best mean scores (i.e., 0.590). The quartile range of FeSCH is smaller than that of Peters, HISNN and TCA+, while Burak has the smallest quartile range.

Based on the above analysis, compared with the four state-of-the-art methods, FeSCH can achieve better performance under CPDP context. The AUC measure normally conforms to the $F1$-measure, which means a method with a good AUC score also has a good score of $F1$-measure. In addition, FeSCH has stable performance on both datasets according to all four performance measures. The reasons that FeSCH performs nearly the best among the methods can be summarized as follows. 1) FeSCH not only considers the distribution differences of features between the source and the target projects, but also takes into account the redundancy among the features. 2) Instance filtering requires large enough source project data, because the number of instances (i.e., the size of training set) is more essential than the number of features in training a high-quality defect predictor. When the source project data are not sufficient, the instance filtering methods may perform poor. 3) Feature mapping performs well when having sufficient source data, but the performance may be sacrificed by information loss during the feature mapping. Above all, feature selection may seem promising under the CPDP context, but extensive studies are still required to make a sound conclusion.

6 Threats to Validity

Threats to the internal validity are that faults may exist in the implementation of the methods. To minimize the internal threats, we implement these methods by pair programming, and make full use of the third-party implementations such as the LIBLINEAR and MINE toolkits. Moreover, the results of the baseline method (e.g., TCA+) are gathered from their published work[1].

Threats to the external validity of our study are that the observed experimental results may not be applicable to other software projects. To guarantee the universality of the experimental results, we choose two extensively used datasets ReLink and AEEEM[1,23,27]. In addition, we choose Logistic Regression as the default classifier, which is most used in recent CPDP research. Other commonly used classifiers (Naive Bayes and Random Forest) are also implemented to reduce the external threats.

Threats to the construct validity are that the performance evaluation may not be representative of the real-world requirements for software defect prediction. To minimize the threats, we use $F1$-measure, which is commonly used in CPDP[1,5] and makes a good balance between the precision and recall measure. During discussion, we add AUC to reinforce the validity of the experimental results. The non-parametric statistical hypothesis test (Wilconxon signed-rank test) is conducted to ensure the confidence of performance comparison among the methods.

7 Conclusions

In this paper, for cross-project defect prediction, we proposed a cluster-based method FeSCH for selecting suitable features to alleviate the distribution gap between the source and the target project data. FeSCH clusters features using a density-based clustering method DPC and selects features from these clusters using one of the three ranking strategies. To evaluate the performance of FeSCH, we designed experiments using real-world project data, and studied the effects of ranking strategies, feature selection ratio, classification models, and feature categories. The experimental results proved that FeSCH can outperform the other baseline methods in most cases and its performance does not rely on the classifiers trained. In addition, guidelines are provided for choosing suitable features, including the optimal ranking strategy, the suitable feature selection ratio, and the favorable software metrics under cross-project software defect prediction.

In the future work, firstly, we plan to collect extra datasets to verify the generality of our empirical results and use other classification models (such as support vector machines and neural networks) to investigate the potential of FeSCH. Secondly, we plan to design new feature selection strategies and clustering methods to improve FeSCH. Thirdly, we will study how to select the optimal design options, such as the feature ratio through self-learning.

References

[1] Nam J, Pan S J, Kim S. Transfer defect learning. In *Proc. the 35th Int. Conf. Software Engineering*, May 2013, pp.382-391.

[2] Zhang F, Keivanloo I, Zou Y. Data transformation in cross-project defect prediction. *Empir. Softw. Eng.*, 2017, 22(6): 3186-3218.

[3] Herbold S. Training data selection for cross-project defect prediction. In *Proc. the 9th Int. Conf. Predictive Models in Software Engineering*, October 2013, Article No. 6.

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

[5] Peters F, Menzies T, Marcus A. Better cross company defect prediction. In *Proc. the 10th Working Conf. Mining Software Repositories*, May 2013, pp.409-418.

[6] Amasaki S, Kawata K, Yokogawa T. Improving cross-project defect prediction methods with data simplification. In *Proc. the 41st Euromicro Conf. Software Engineering and Advanced Applications*, August 2015, pp.96-103.

[7] Rodriguez A, Laio A. Clustering by fast search and find of density peaks. *Science*, 2014, 344(6191): 1492-1496.

[8] Briand L C, Melo W L, Wust J. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Trans. Softw. Eng.*, 2002, 28(7): 706-720.

[9] 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 *Proc. the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. the Foundations of Software Engineering*, August 2009, pp.91-100.

[10] He Z M, Shu F D, Yang Y, Li M S, Wang Q. An investigation on the feasibility of cross-project defect prediction. *Autom. Softw. Eng.*, 2012, 19(2): 167-199.

[11] Ma Y, Luo G C, Zeng X, Chen A G. Transfer learning for cross-company software defect prediction. *Inf. Softw. Technol.*, 2012, 54(3): 248-256.

[12] Ryu D, Jang J I, Baik J. A hybrid instance selection using nearest-neighbor for cross-project defect prediction. *J. Comput. Sci. Technol.*, 2015, 30(5): 969-980.

[13] Herbold S, Trautsch A, Grabowski J. Global vs. local models for cross-project defect prediction. *Empir. Softw. Eng.*, 2017, 22(4): 1866-1902.

[14] Wang S, Liu T Y, Tan L. Automatically learning semantic features for defect prediction. In *Proc. the 38th Int. Conf. Software Engineering*, May 2016, pp.297-308.

[15] Chen L, Fang B, Shang Z W, Tang Y Y. Negative samples reduction in cross-company software defects prediction. *Inf. Softw. Technol.*, 2015, 62: 6777.

[16] Canfora G, De Lucia A, Di Penta M, Oliveto R, Panichella A, Panichella S. Multi-objective cross-project defect prediction. In *Proc. the 6th Int. Conf. Software Testing Verification and Validation*, March 2013, pp.252-261.

[17] Panichella A, Oliveto R, De Lucia A. Cross-project defect prediction models: L'union fait la force. In *Proc. Conf. Software Maintenance Reengineering and Reverse Engineering*, February 2014, pp.164-173.

[18] Zhang Y, Lo D, Xia X, Sun J L. An empirical study of classifier combination for cross-project defect prediction. In *Proc. the 39th Annual Computer Software and Applications Conf.*, July 2015, 2: 264-269.

[19] Zhang F, Mockus A, Keivanloo I, Zou Y. Towards building a universal defect prediction model. In *Proc. the 11th Working Conf. Mining Software Repositories*, May 2014, pp.182-191.

[20] Xia X, Lo D, Pan S J, Nagappan N, Wang X Y. HYDRA: Massively compositional model for cross-project defect prediction. *IEEE Trans. Softw. Eng.*, 2016, 42(10): 977-998.

[21] Herbold S. CrossPare: A tool for benchmarking cross-project defect predictions. In *Proc. the 30th ACM/IEEE Int. Conf. Automated Software Engineering Workshop* November 2015, pp.90-96.

[22] Nam J, Kim S. Heterogeneous defect prediction. In *Proc. the 10th Joint Meeting on Foundations of Software Engineering*, September 2015, pp.508-519.

[23] Jing X Y, Wu F, Dong X W, Qi F M, Xu B W. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In *Proc. the 10th Joint Meeting on Foundations of Software Engineering*, August 30-September 4, 2015, pp.496-507.

[24] Kamei Y, Fukushima T, McIntosh S, Yamashita K, Ubayashi N, Hassan A E. Studying just-in-time defect prediction using cross-project models. *Empir. Softw. Eng.*, 2016, 21(5): 2072-2106.

[25] Hosseini S, Turhan B, Mäntylä M. Search based training data selection for cross project defect prediction. In *Proc. the 12th Int. Conf. Predictive MODELS and Data Analytics in Software Engineering*, September 2016, Article No. 3.

[26] Nam J, Kim S. CLAMI: Defect prediction on unlabeled datasets. In *Proc. the 30th ACM/IEEE Int. Conf. Automated Software Engineering*, November 2015, pp.452-463.

[27] Zhang F, Zheng Q, Zou Y, Hassan A E. Cross-project defect prediction using a connectivity-based unsupervised classifier. In *Proc. the 38th Int. Conf. Software Engineering*, May 2016, pp.309-320.

[28] Gao K H, Khoshgoftaar T M, Wang H J, Seliya N. Choosing software metrics for defect prediction: An investigation on feature selection techniques. *Softw.: Pract. Exper.*, 2011, 41(5): 579-606.

[29] Shivaji S, Whitehead E J, Akella R, Kim S. Reducing features to improve code change-based bug prediction. *IEEE Trans. Softw. Eng.*, 2013, 39(4): 552-569.

[30] Xu Z, Liu J, Yang Z J, An G G, Jia X Y. The impact of feature selection on defect prediction performance: An empirical comparison. In *Proc. the 27th IEEE Int. Symp. Software Reliability Engineering*, October 2016, pp.309-320.

[31] Xu Z, Liu J, Xia Z, Yuan P P. An empirical study on the equivalence and stability of feature selection for noisy software defect data. In *Proc. the 29th Int. Conf. Software Engineering and Knowledge Engineering*, July 2017, pp.191-196.

[32] Ghotra B, McIntosh S, Hassan A E. A large-scale study of the impact of feature selection techniques on defect classification models. In *Proc. the 14th Int. Conf. Mining Software Repositories*, May 2017, pp.146-157.

[33] Liu S L, Chen X, Liu W S, Chen J Q, Gu Q, Chen D X. FECAR: A feature selection framework for software defect prediction. In *Proc. the 38th Annual Computer Software and Applications Conf.*, July 2014, pp.426-435.

[34] Liu W S, Liu S L, Gu Q, Chen J Q, Chen X, Chen D X. Empirical studies of a two-stage data preprocessing approach for software fault prediction. *IEEE Trans. Reliab.*, 2016, 65(1): 38-53.

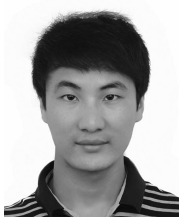
[35] Liu W S, Chen X, Gu Q, Liu S L, Chen D X. A cluster-analysis-based feature-selection method for software defect prediction. *Sci. Sin. Inf.*, 2016, 46(9): 1298-1320.

[36] Reshef D N, Reshef Y A, Finucane H K, Grossman S R, McVean G, Turnbaugh P J, Lander E S, Mitzenmacher M, Sabeti P C. Detecting novel associations in large data sets. *Science*, 2011, 334(6062): 1518-1524.

[37] Fan R E, Chang K W, Hsieh C J, Wang X R, Lin C J. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.*, 2008, 9: 1871-1874.

[38] Wu R X, Zhang H Y, Kim S, Cheung S C. ReLink: Recovering links between bugs and changes. In *Proc. the 19th ACM SIGSOFT Symp. and the 13th European Conf. Foundations of Software Engineering*, September 2011, pp.15-25.

[39] D'Ambros M, Lanza M, Robbes R. An extensive comparison of bug prediction approaches. In *Proc. the 7th IEEE Working Conf. Mining Software Repositories*, May 2010, pp.31-41.



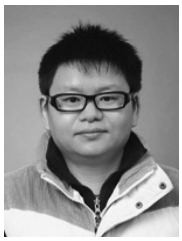
Chao Ni received his B.S. degree in computer science from Nantong University, Nantong, in 2014. Then he received his M.S. degree in computer science from Nanjing University, Nanjing, in 2017. Now he is a Ph.D. candidate of State Key Laboratory for Novel Software Technology and the Department of Computer Science and Technology, Nanjing University, Nanjing. His research interests are mainly in software defect prediction and machine learning.

Wang-Shu Liu received his B.S. and M.S. degrees in computer science from Nanjing University of Science and Technology, Nanjing, in 2010 and 2013, respectively. He is now a Ph.D. candidate of State Key Laboratory for Novel Software Technology and the Department of Computer Science and Technology, Nanjing University, Nanjing. His research interests include software defect prediction and machine learning.

Xiang Chen received his B.S. degree in the School of Management from Xi'an Jiaotong University, Xi'an, in 2002. Then he received his M.S. and Ph.D. degrees in computer science from Nanjing University, Nanjing, in 2008 and 2011, respectively. Now he is an associate professor in the School of Computer Science and Technology, Nantong University, Nantong. His research interests are mainly in software testing, such as software defect prediction, combinatorial testing, regression testing, and software fault localization. He has published over 40 papers in referred journals and conferences.

Qing Gu received his Ph.D. degree in computer science from Nanjing University, Nanjing. He is a professor of the State Key Laboratory of Novel Software Technology, and the Department of Computer Science and Technology, Nanjing University, Nanjing. His research interests include software testing, quality and process improvement, software maintenance and evolution, and complex network.

Dao-Xu Chen is currently a full professor of the State Key Laboratory of Novel Software Technology and the Department of Computer Science and Technology, Nanjing University, Nanjing. His research interests include distributed computing, parallel processing, and computer networks. He is a fellow of CCF and a member of ACM and IEEE.



Qi-Guo Huang received his Bachelor of Arts in English in the School of Foreign Language from Nanjing University of Finance & Economics, Nanjing, in 2008. Then he received his M.S. degree from the School of Software from Nanjing University, Nanjing, in 2012. Now he is a Ph.D. candidate at the Department of Computer Science and Technology and the State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing. His research interests are mainly in big data analysis for software engineering.