Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Automated change-prone class prediction on unlabeled dataset using unsupervised method

Meng Yan[b], Xiaohong Zhang[a,b,*], Chao Liu[b], Ling Xu[b], Mengning Yang[b], Dan Yang[b]

[a] *Key Laboratory of Dependable Service Computing in Cyber Physical Society Ministry of Education, Chongqing 400044, PR China*
[b] *School of Software Engineering, Chongqing University, Chongqing 401331, PR China*

ARTICLE INFO

Article history:
Received 20 September 2016
Revised 17 February 2017
Accepted 4 July 2017
Available online 6 July 2017

Keywords:
Software maintenance
Change-prone prediction
Unlabeled dataset
Unsupervised prediction

ABSTRACT

Context: Software change-prone class prediction can enhance software decision making activities during software maintenance (e.g., resource allocating). Researchers have proposed many change-prone class prediction approaches and most are effective on labeled datasets (projects with historical labeled data). These approaches usually build a supervised model by learning from historical labeled data. However, a major challenge is that this typical change-prone prediction setting cannot be used for unlabeled datasets (e.g., new projects or projects with limited historical data). Although the cross-project prediction is a solution on unlabeled dataset, it needs the prior labeled data from other projects and how to select the appropriate training project is a difficult task.
Objective: We aim to build a change-prone class prediction model on unlabeled datasets without the need of prior labeled data.
Method: We propose to tackle this task by adopting a state-of-art unsupervised method, namely CLAMI. In addition, we propose a novel unsupervised approach CLAMI+ by extending CLAMI. The key idea is to enable change-prone class prediction on unlabeled dataset by learning from itself.
Results: The experiments among 14 open source projects show that the unsupervised methods achieve comparable results to the typical supervised within-project and cross-project prediction baselines in average and the proposed CLAMI+ slightly improves the CLAMI method in average.
Conclusion: Our method discovers that it is effective for building change-prone class prediction model by using unsupervised method. It is convenient for practical usage in industry, since it does not need prior labeled data.

1. Introduction

Software maintenance has been regarded as one of the most expensive and tough tasks in the whole software lifecycle [1]. Change is fundamental for software maintenance according to the technological advancements and new requirements. Managing and controlling change in software maintenance is one of the significant concerns of the software industry [2]. A change could be made because of existence of bugs, new features or refactoring [3,4]. It is the source of defects and modifications. Understanding the knowledge about change-prone classes in a software is significant for developers and mangers [5]. A change-prone class means that the class is likely to change with a high probability after a product release. It can represent the weak part of a software system [2]. Thus, software change-prone class prediction contributes to better allocation of software resources (e.g., time and staff) in the software maintenance process [6]. This technique aids to support maintenance related decision making by identifying change-prone classes in advance. As a result, the quality assurance teams or testers can determine the critical parts of the software where the quality assurance or testing activities should pay more attention and track rigorously.

In order to predict change-prone classes in advance, various categories of software metrics which indicate various characteristics have been proved to correspond to the change-proneness, such as OO metrics (e.g., cohesion, coupling, inheritance, etc.) [7], code smells [8], design patterns and [9] evolution metrics [10,11]. Based on these metrics, a number of studies which use machine learning techniques have been proposed for building change-prone class prediction models, such as Bayesian networks [12], neural networks [13], and ensemble methods [6]. A typical prediction model based on machine learning is designed by learning from historical labeled data within a project in a supervised way as Fig. 1(a) shows. This manner is referred as supervised within-project pre-

[*] Corresponding author at: School of Software Engineering, Chongqing University, Huxi Town, Shapingba, Chongqing, PR. China 401331.
E-mail address: xhongz@cqu.edu.cn (X. Zhang).

http://dx.doi.org/10.1016/j.infsof.2017.07.003
0950-5849/© 2017 Elsevier B.V. All rights reserved.

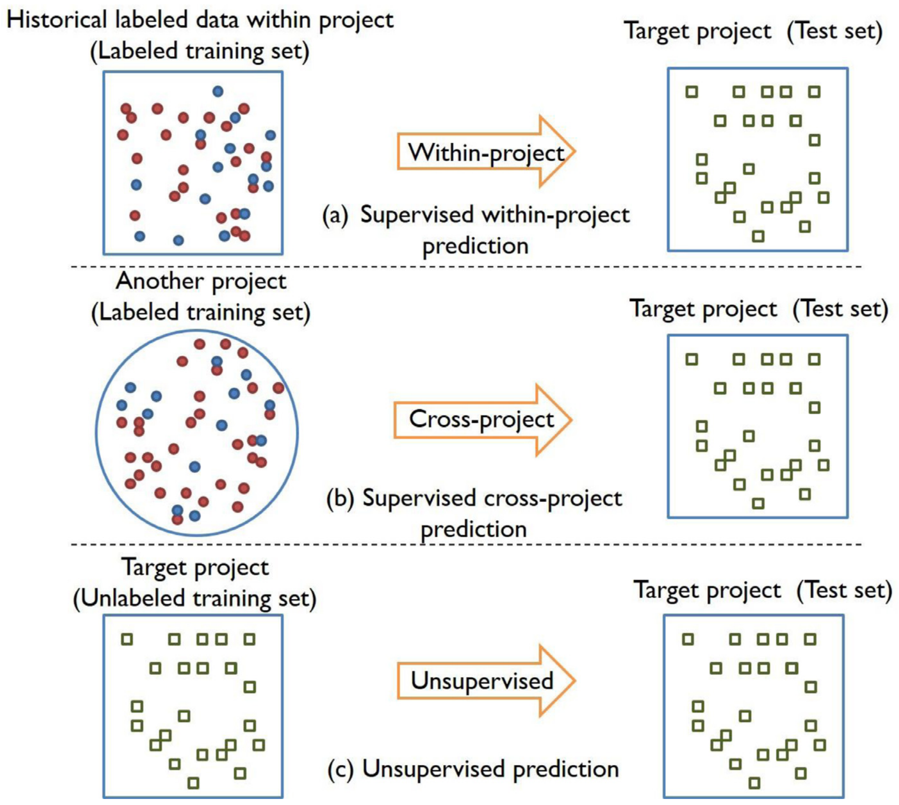

Historical labeled data within project
(Labeled training set)
Within-project
Target project (Test set)
(a) Supervised within-project prediction
Another project
(Labeled training set)
Cross-project
Target project (Test set)
(b) Supervised cross-project prediction
Target project
(Unlabeled training set)
Unsupervised
Target project (Test set)
(c) Unsupervised prediction

Fig. 1. Illustration of three prediction manners. Manner (a) is the supervised within-project prediction which training on historical labeled data and testing on target data within a project. Manner (b) is the cross-project prediction which training on another labeled project data and testing on the target project. Manner (c) is the unsupervised prediction which directly learning from the target project data.

diction [14]. Namely, the key idea is to train the model on historical labeled data within a project and then predict the target data. We refer the dataset which have historical labeled data as "*labeled dataset*". However, in practice, it is often time-consuming and expensive to collect labeled data. Furthermore, this manner is difficult to apply on new projects or projects with limited historical data whose label information are unavailable *(referred to as "unlabeled dataset"),* since it is difficult to collect label information for training a prediction model.

Cross-project change-prone class prediction method has been proposed to address the above-mentioned issue [14] as Fig. 1(b) shows. The cross project technique is motivated by the similar techniques in defect prediction [15,16]. It enables change-prone class prediction on unlabeled projects by learning from other projects which are already labeled. However, one issue which remains is that training set and testing set in cross-project prediction come from different project which possess different distributions [17]. The distribution similarity of training set and testing set is important for building a prediction model [18,19]. As a result, the success rate (ratio of combination whose performance is greater than a certain threshold) of cross-project prediction reported in the work [14] is generally poor (30%). Moreover, the cross-project change-prone prediction may not be effective and it depends on the selection of the source project [14].

To address the above-mentioned limitation, we propose to tackle this task by using unsupervised method as Fig. 1(c) shows. Compared with supervised models, unsupervised method does not need the prior labeled data to build prediction models which are more desirable in practice. It has been widely used in software quality prediction [17,20,21]. In detail, we apply a state-of-art unsupervised method (CLAMI: Clustering, Labeling, Metric selection and Instance selection) to the change-prone class prediction which has been successfully used in another field [17]. The key idea is to conduct the prediction on unlabeled dataset by learning from itself. Strictly, it is a special case of within-project manner. In this work, we use unsupervised refers to as the prediction without the need of historical labeled data particularly. Concretely, clustering is to group the instances, labeling is to estimate the label of groups by using a unsupervised way, metric selection and instance selection is to select more informative features and training sets. By the following, we predict the target set by training on the selected features and training sets.

The detailed process of the unsupervised method can be interpreted by dividing three phases as Fig. 2 shows. Each phase consists of two steps. The clue of the whole process is to build the prediction model by learning on selected informative metrics and instances from the target dataset itself. In detail, the first phase is clustering and labeling. In this phase, an unlabeled dataset is clustered into groups according to the difference between metric value and metric threshold. Subsequently, we estimate the labels of the dataset according to the magnitude of metric values [17]. The goal of this phase is to provide the estimated labels of all the instances. However, the estimated labels of all the instances might not be correct enough. In our unsupervised method, part of them will be automatically selected as final training set according to our criteria in the following phase. The second phase is to conduct the metric selection and instance selection from the labeled instances in the first phase. As a result, an informative training set of metrics and instances are generated. The third phase is modeling and prediction. The prediction model is built by learning from the selected instances and features in the second phase.

In particular, the labeling step in the first phase of the adopted method CLAMI is conducted by measuring the count of violation (i.e., a metric value is greater than a certain threshold) of an in-

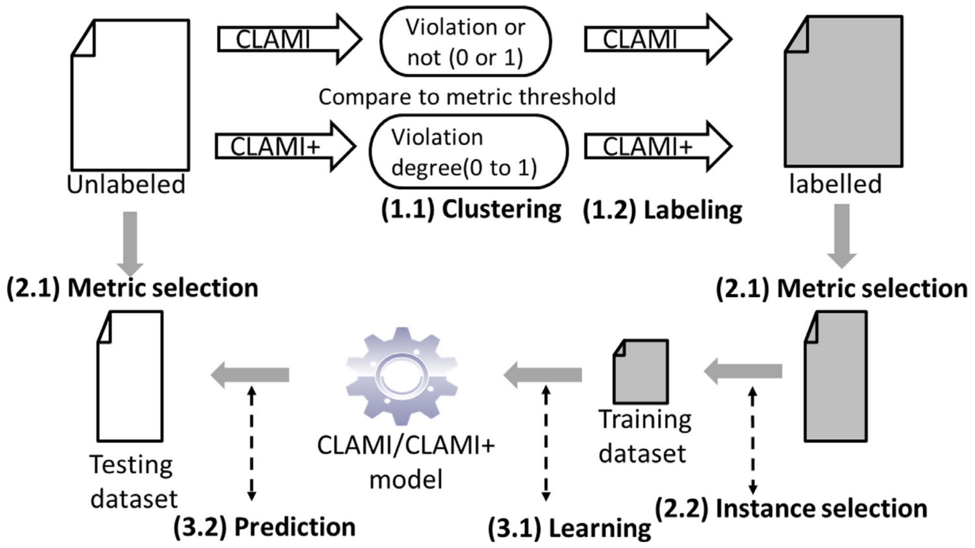


Fig. 2. The overview of CLAMI/CLAMI+. It consists of three phases with two steps in each phase. The first phase is clustering and labeling (step 1.1 and 1.2), the CLAMI and CLAMI+ are different in this phase. The second phase is metric selection and instance selection (step 2.1 and 2.2). The third phase is learning and prediction (step 3.1 and 3.2).

stance. However, we observe that information loss might result from mapping the violation to a 1 or 0 (i.e., violation or not) result in the first phase. The information that how much the instance violated on a metric is not considered. Based on this observation, we propose a novel unsupervised method CLAMI+ by extending CLAMI. The difference lies in the first phase as Fig. 2 shows. In detail, the CLAMI+ method uses the violation degree (i.e., transforming the difference between the metric value and the threshold to a probabilistic value) to replace the Boolean representation in CLAMI. As a result, the fine information that how much the instance violated on a metric is considered. Under this way, the selection of final training set of CLAMI+ is different from CLAMI. The training set generated by CLAMI+ is expected more informative that CLAMI which is beneficial for building prediction model.

The goal of our study is to conduct the change-prone class prediction on unlabeled dataset in an automated way without the need of historical data. In our empirical experiments, we evaluate the unsupervised methods on 14 open source projects which come from the Qualitas Corpus [22]. As a result, the unsupervised methods yields a reasonable performance which improves the typical within-project prediction models by 5.2%–19.7% (average CCR), 13.9%–27.2% (average AUC) and 21.7%–61.9% (average F-measure), respectively. Meanwhile, considering the comparison with cross-project method, the CLAMI and CLAMI+ methods improve it by 5.4% and 7.9% (average CCR), 14.5% and 18.2% (average AUC), 35.1% and 40.5% (average F-measure), respectively. In addition, considering the comparison between CLAMI and CLAMI+, the proposed CLAMI+ improves the CLAMI by 2.9% (average CCR), 3.2% (average AUC) and 4% (average F-measure).

This paper extends our preliminary study [23] published as a conference paper. In summary, the main extensions are as follows: (1) additional comparison of unsupervised method over cross-project method is added, since both of them are suitable on unlabeled datasets (see Section 5.2). (2) Exploration on the effect of varying the steepness of sigmoid function to the performance of our method is added (see Section 5.4). (3) Exploration on the effect of varying the metric threshold to the performance of our method is added (see Section 5.5). (4) More detailed descriptions of the motivation, related work, and approach are added. In addition, more expanded and qualitative analysis about the results are added.

In summary, the contributions of this study are as follows:

- We apply unsupervised approach to tackle change-prone class prediction on new projects or projects with limited historical data. In addition, we propose a novel unsupervised method CLAMI+ by extending the method CLAMI. (See Section 3)
- We present an empirical study to evaluate the unsupervised methods compared with supervised within-project and cross-project methods on 14 public datasets. The results show that the unsupervised methods achieve a reasonable performance which outperform the baselines in average. (See Sections 5.1 and 5.2)
- We explore the effect of two factors to the performance of our proposed method: the metric threshold and a transformation parameter in the first phase. As a result, we find that the two factors impact the performance of our method. (See Sections 5.4 and 5.5)

This article is structured as follows: in Section 2, we describe the related work. Section 3 shows the detailed description of our approach. In Section 4, we provided the experimental design of the empirical study, including the research questions, datasets, baselines and measures. Next, Section 5 reports the experimental results and the answers for each research question. In Section 6, we provide the threats to validity of this work. Finally, we draw our conclusions and the future work plans in Section 7.

2. Related work

2.1. Change-prone prediction on labeled dataset

A variety of approaches have been proposed to predict change-prone classes by learning from historical labeled dataset. For example, Amoui et al [13] proposed an innovative Neural Network-based temporal change prediction model which can predict where and when the change will happen. They achieved a reasonable performance on Mozilla and Eclipse. Giger et al [24] proposed a Neural Network model to identify potentially change-prone classes by using metrics from social network analysis (SNA) and OO metrics. The results show that the Neural Networks model based on SNA and OO metrics outperforms the model based on SNA or OO metrics separately. Godara and Singh [25] proposed an ID3 prediction model based on multi-factors, including trace events generated from code, execution time and behavioral dependency. Koru and Liu [26] first validated the Pareto's law on change-prone classes on two open source projects, namely KOffice and Mozilla. They found the applicability of Pareto's law and developed a tree based prediction model. Lu et al [27] proposed a statistical meta-analysis approach to explore the ability of 62 OO metrics for predicting change-proneness on 102 Java systems. They found that size metrics were more discriminative that other OO metrics, such as cohesion, coupling and inheritance. Elish et al [6] proposed an empirical study which used ensemble methods on change prediction. The base learners adopted in their work are typical machine learning methods, such as Multilayer perceptron, Radial basis function network, Support vector machine, and Logistic regression. They found that ensemble methods can achieve a better performance than using individual models. However, one issue in the above-mentioned works is that the prediction model relies on learning from the historical labeled dataset in a supervised way. It is difficult to apply the technique on unlabeled dataset.

2.2. Change-prone prediction on unlabeled dataset

Cross-project prediction is a solution to address the above-mentioned limitation. The cross-project concept is introduced by Briand et al [28]. The basic thinking is to train the prediction model by using labeled dataset from another project. It has been widely used in defect prediction [15,29,30,31,32]. In change-prone class prediction, there are also a few studies which have investigated the cross-project change prediction recently.

Malhotra and Bansal [14] proposed to build the cross project change prediction model by using the logitboost method. In another work, Malhotra and Bansal [33] validated the cross project change prediction by using machine learning and search-based techniques. However, they found that the cross-project (or inter-project) prediction cannot comparable to the within-project prediction [14]. Besides, one main issue remains in cross-project prediction is that different projects possess different data distributions [17]. How to select an appropriate source as the training data is a difficult task [14].

To address the above-mentioned limitation, the unsupervised method can enable the prediction task which does not need a prior labeled source as the training dataset.

3. Approach

This section describes the process of the proposed unsupervised approach CLAMI+ which extends the state-of-art method CLAMI. It consists of three phases and we describe the three phases in three subsections (Sections 3.1, 3.2 and 3.3). The first phase is Clustering and Labeling, the second phase is Metric selection and Instance selection, the third phase is Learning and Prediction. In particular, the idea of the first phase in CLAMI+ is different with CLAMI, and the idea of the second phase and the third phase in CLAMI+ is identical with CLAMI.

3.1. Clustering and labeling

(1) *Clustering*: The goal of this step is to clutter all the instances according to the difference between metric value and metric threshold. A threshold means that the instance whose metric values exceed a threshold is more change-prone [34]. In our approach, we set the threshold as the median value as Nam and Kim [17] described, and we apply various cutoff thresholds to explore the effect to the performance in Section 5.5. Fig. 3 illustrates the clustering process in the unsupervised approach. We use A-G to denote the instances of the dataset and $X_1 - X_7$ denote the adopted metrics. The first step is to compare the metric value to the threshold value for each metric. The higher metric values which exceed the threshold are in bold. Subsequently, we transform the difference between metric value and metric threshold to a violation value which indicates whether (CLAMI) or how much (CLAMI+) does the metric value exceeds the threshold. As a result, a violation table is generated.

In CLAMI, the violation table consists of 0 or 1 values. The value "1" represents a higher value which means it is greater than the threshold value as highlighted in Fig. 3. After that, the clustering process groups the instances by the sum of the count of the higher metric values (refered to as K value). For example, the instance A and E belong to one cluster ($K = 3$) which means there are three higher metric values in A and E. However, one issue remains in CLAMI is that the information that how much the instance violated on a metric is not considered. For example, considering instance B and E at the metric X_6, the metric value of B is 3 and the metric value of E is 10. Although both of them are higher values which is greater than threshold 1, the violation degree of instance E is greater thant instance B obviously. This information is ignored in CLAMI.

In CLAMI+, we extend the CLAMI approach by transforming the 1 or 0 result (violation or not) to a continuous value from 0 to 1 which represents the violation degree. As a result, the violation table consists of continuous values ranging from 0 to 1 as Fig. 3 shows. In detail, we adopt the sigmoid function which is often used as the activation function in neural networks to conduct the transformation. Formally, suppose there are M instances and N metrics, Xij denotes the jth metric value of the i-th instance, Nj denotes the threshold value of the jth metric. The violation degree of the jth metric of the ith instance $P(Vij)$ is computed as Formula (1). Different from the CLAMI, the K value in CLAMI+ represents the mean violation degree of an instance and we cluster the instances into two clusters by $K >= 0.5$ and $K < 0.5$.

$$P(V_{ij}) = \frac{1}{1 + e^{-(X_{ij} - N_j)}} \tag{1}$$

(2) *Labeling*: In CLAMI, the labeling step is conducted by dividing the clusters into a top half and a bottom half by considering the K value [17]. Next, the first half clusters are labeled as change-prone and the bottom half clusters are labeled as not change-prone. Similar to CLAMI, in CLAMI+, we label the instances by dividing the clusters into $K >= 0.5$ and $K < 0.5$. Next, we label the first cluster as change-prone and the second cluster as not change-prone. As the Fig. 3 shows, in CLAMI, Instance C, A and E are labeled as change-prone while in CLAMI+ Instance A, B, C, E and F are labeled as change-prone. This difference is resulted from our usage of the violation degree.

The labeling step is motivated by the tendency in defect prediction, namely, the defect-prone instances have higher metric values than clean-prone instances [17,35]. Since the typical metrics which are usually adopted in both defect prediction and change prediction (e.g., OO metrics and typical size metrics) represent the complexity of the instance, there is also the similar tendency in change-prone prediction [34,36] (named as change-prone tendency). For example, Koru and Tian [36] found that high-change modules had fairly high places in metric rankings, although not the highest places. Malhotra and Bansal [34] found that the classes whose metric values exceed a threshold value are change prone.

3.2. Metric selection and instance selection

In order to generate a high-quality training set, we use metric and instance selection to select more informative metrics and minimize the instances that may be incorrectly labeled in the first phase.

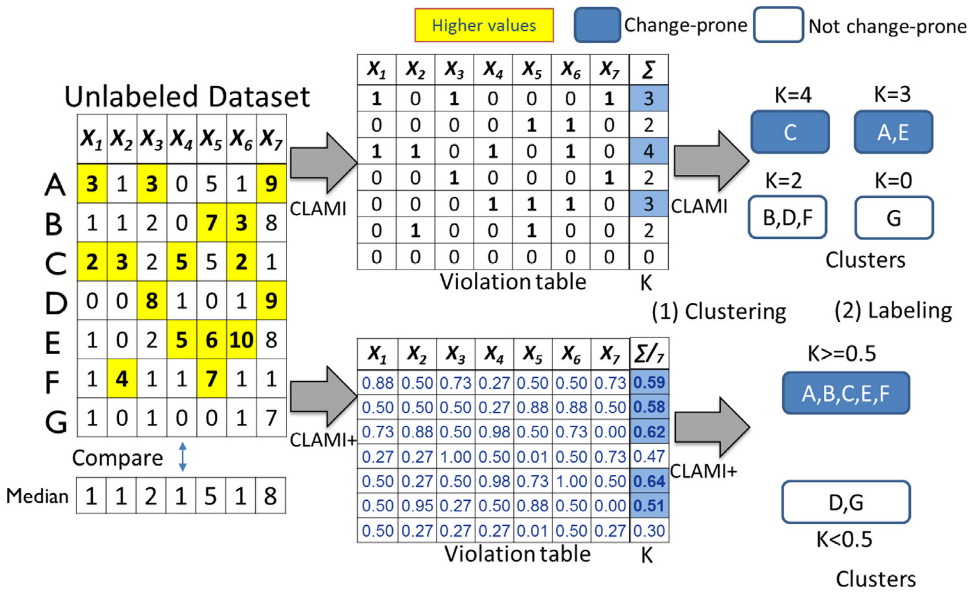


Fig. 3. Illustration of the first phase in the unsupervised approach.

(1) *Metric selection*: The quality of features plays a significant role in building a prediction model. Since there might be some metrics which do not follow the change-prone tendency well, the objective of metric selection step is to select the most informative metrics which can enhance the prediction ability. The selection criteria is the metric violation scores (MVS) for each metric. A violation means the metric value does not follow the change-prone tendency. In terms of one metric, the MVS is equal to the count of instances which do not follow the change-prone tendency on this metric. Take the metric X_1 in Fig. 3 as the example, instance B is labeled as a change-prone instance in the first phase of CLAMI+, however, the metric value of X_1 is not a higher value, thereby B does not follow the tendency at metric X_1. Using this way, we compute the MVS for each metric and select the metrics which have the minimum MVS.

(2) *Instance Selection*: In order to generate a better training set, instance selection is a widely adopted technique in software prediction models [37,38]. It is the final step for generating the training dataset in this unsupervised method. In detail, we select the instances which follow the change-prone tendency at the selected metrics. In other words, we remove the instances which do not follow the change-prone tendency on the selected metrics. For example, suppose X_1 is a selected metric in Fig. 3 and B is labeled as a change-prone instance in the first phase of CLAMI+. However, the metric value of X_1 does not follow the change-prone tendency (the metric value is expected to greater than threshold) in Instance B, thereby we will remove B from the final training set. After this step, in some cases which have too many tendency-violated instances, there might be no change-prone or not change-prone instances. In this sense, we will get back to the metric selection step and choose extra metrics which have the next minimum MVS until both change-prone and not change-prone instances exist in the training set.

3.3. Learning and prediction

After generating a training set, we adopt a general machine learner (logistic regression) to build the prediction model which learns from the selected metrics and instances. By the following, we predict the change-prone classes of the testing set on the selected metrics.

4. Experimental design

4.1. Research questions

We design five research questions (RQs) to evaluate this study. RQ1 is designed to evaluate the performance of unsupervised method compared with the within-project prediction method including four supervised methods and one typical clustering method. RQ2 is designed to compare the performance of unsupervised method with the cross-project prediction method. RQ3 is designed to evaluate the extended CLAMI+ by comparing with the adopted CLAMI. RQ4 is to explore the effect of the parameter of the adopted sigmoid function to the prediction performance. RQ5 is to explore the effect of different thresholds.

- **RQ1:** Is the prediction performance of the unsupervised method comparable to typical within-project prediction methods?

We will answer this question by comparing the unsupervised method and the supervised within-project prediction methods on the same target dataset. The difference is that the supervised within-project prediction methods learn from historical labeled data while our unsupervised method does not need prior labeled data.

- **RQ2:** Is the prediction performance of the unsupervised method comparable to cross-project prediction method?

Building cross-project prediction model is also an alternative solution to predict unlabeled project when we have access to another labeled project. Since both unsupervised and cross-project methods aim at building prediction model for unlabeled dataset, this research question is designed to compare the performance of two solutions on the same target datasets.

- **RQ3:** Does the prediction performance of CLAMI+ outperform CLAMI?

The difference between CLAMI and CLAMI+ is the criteria of clustering and labeling. As a result, the training set is different which has an impact on the prediction performance. We will answer this question by comparing the two methods on the same datasets.

- **RQ4:** What is the effect of varying the steepness of sigmoid function to the performance of CLAMI+?

Table 1
Summary of the evaluation datasets in this study.

Previous version	Target version	% changed	# instances
‘ant-1.8.1.0’	‘ant-1.8.2.0’	12.20%	844
‘antlr-3.3.0’	‘antlr-3.4.0’	70.95%	241
‘argouml-0.32.1’	‘argouml-0.32.2’	39.67%	1505
‘azureus-4.1.0.2’	‘azureus-4.1.0.4’	7.71%	3150
‘freecol-0.10.4’	‘freecol-0.10.5’	71.74%	598
‘freemind-0.6.5’	‘freemind-0.6.7’	89.19%	74
‘hibernate-3.1.1.0’	‘hibernate-3.1.2.0’	93.62%	925
‘jgraph-5.12.0.4’	‘jgraph-5.12.1.0’	20.75%	53
‘jmeter-2.7.0.0’	‘jmeter-2.8.0.0’	58.07%	830
‘jstock-1.0.7.1’	‘jstock-1.0.7.2’	11.59%	276
‘jung-1.7.2’	‘jung-1.7.4’	28.85%	468
‘junit-4.9.0’	‘junit-4.10.0’	92.02%	163
‘lucene-3.6.2.0’	‘lucene-4.0.0.0’	34.35%	620
‘weka-3.5.7’	‘weka-3.5.8’	13.94%	1119

In the CLAMI+ method, we adopt the sigmoid function to perform the transformation task, namely, transform the difference between the metric value and the metric threshold to the violation degree. The sigmoid function is a special case of logistic function. The equation in our case is as formula (2) shows. The L represents the maximum value and S represents the steepness of the function curve. In our case, the maximum value represents a probabilistic value which is fixed at $L = 1$. However, the steepness of the function curve represents how sensitive of the transformation with the difference between the metric value and the metric threshold. In our previous work, we use the default setting: $S = 1$. In order to investigate the effect of steepness to the performance, we will answer this question by varying the steepness in a certain range.

$$P(V_{ij}) = \frac{L}{1 + e^{-S(X_{ij} - N_j)}} \tag{2}$$

- **RQ5**: Which is the effect of varying the metric threshold to the performance of CLAMI+?

The identification of the higher metric values relies on the choice of the threshold. Different thresholds can result in different clustering and labeling results which have an impact on the prediction model. This research question is designed to explore what the effect of varying the metric thresholds. We will apply different thresholds on the experiments to analysis the effect to the performance.

4.2. Datasets

We evaluate this study on 14 open source projects which come from the public dataset Qualitas Corpus [22] (Qualitas Corpus version is 20130901e). They are written in Java and have multiple evolution versions. For each project, we choose the recent version as the target dataset and label each instance by tracking the version control system. The target project version, previous version (only used in baselines), percentage of changed instances and the total number of instances in the target version are listed in Table 1. The percentage and the number of instances possess a substantial range which can validate the model ability among a wide range.

Considering the code metrics, we adopt the typical metrics which are identical with the relevant studies of change prediction [1,6,12,39] as the Table 2 shows. In detail, five Chidambar and Kemerer metrics [40]: WMC, DIT, NOC, RFC, and LCOM; four Li and Henry metrics [41]: MPC, DAC, NOM, SIZE2; and one traditional lines of code metric (SIZE1) are adopted. SIZE1 represents the number of lines of code excluding comments and SIZE2 represents the total count of the number of data attributes and the number of local methods in a class.

4.3. Experimental baselines

In RQ1, we set the four typical supervised within-project prediction methods and one typical unsupervised clustering method as baselines to compare with our unsupervised methods CLAMI/CLAMI+. In supervised within-project prediction, the basic manner is that in order to predict the change-prone classes of the target data, the prediction models are built by learning from the historical labeled data within the project. In our experiment, the target data is as Table 1 shows (i.e., target version) and we set the previous neighbor version of the target data as the training source in the baselines. In detail, the first baseline is the change-prone class prediction based on logitboost (LB) which is proposed by Malhotra and Bansal [14]. In addition, to avoid the bias from only one classifier, we also adopt three typical machine learners as baselines which are used in all three empirical studies on change prediction proposed by Elish et al [6]. Namely, the second, third and the fourth baseline is Multilayer perceptron (MLP), Radial basis function network (RBF) and Support vector machine (SVM), respectively. At last, since our method is an unsupervised method, we also adopt a typical unsupervised machine learning clustering algorithm, i.e., simple k-means (SK) [42], as the fifth baseline. When implementing the SK method, we set the clusters as 2 and determine which cluster is change-prone by using a heuristic method. In detail, we use the average row sums of the normalized metrics of each cluster to determine which cluster is change-prone as Zhang et al suggested [43]. In RQ2, we set the cross-project change-prone work proposed by Malhotra and Bansal [14] as the baseline to compare with the unsupervised methods. In RQ3, we compare the extended CLAMI+ to the original CLAMI method proposed by Nam and Kim [17].

4.4. Performance measures

Following the similar work of Elish et al [6], two widely used prediction measures are adopted in our evaluation, namely correct classification rate (CCR) and the area under curve (AUC). CCR represents the ratio of cases which were correctly predicted to the total number of cases. AUC represents the area under the receiver operating characteristic (ROC) curve. In addition, we adopt a typical classification performance measure (i.e., F-measure) as the third performance measure. It is a widely used prediction measure since it represents the harmonic mean of precision and recall [17].

5. Results

5.1. Answer of RQ1: performance of the unsupervised and within-project method

Table 3 shows the performance comparison between the unsupervised methods and the baselines in CCR, AUC and F-measure under 14 datasets. In terms of each dataset, if the performance of the unsupervised methods CLAMI/CLAMI+ outperforms all the baselines, the results are bold. The better results between CLAMI and CLAMI+ are underlined.

In summary, in terms of RQ1, considering the CCR measure, the unsupervised methods CLAMI/CLAMI+ outperform the five baselines among 8 datasets and improve them by 5.2%–19.7% in average of all datasets. Considering the AUC measure, CLAMI/CLAMI+ outperform five baselines among 8 datasets and improve them by 13.9%–27.2% in average of all datasets. Considering the F-measure, CLAMI/CLAMI+ outperform five baselines among 7 datasets and improve them by 21.7%–61.9% in average of all datasets. Nonetheless, in some datasets, the proposed unsupervised methods CLAMI/CLAMI+ may perform worse, such as ‘ant-1.8.2.0’, ‘azureus-4.1.0.4’ and ‘jstock-1.0.7.2’. One potential impact factor is the per-

Table 2
Summary of the adopted metrics in this study.

Metric	Description
WMC	Weighted methods per class
DIT	Level for a class within its class hierarchy
NOC	Number of immediate subclasses of a class
RFC	Count of methods implemented within a class plus the number of methods accessible to an object class due to inheritance
LCOM	Counts the sets of methods in a class that are not related through the sharing of some of the class's fields
MPC	The number of messages sent out from a class
DAC	The number of instances of another class declared within a class
NOM	The number of methods in a class
SIZE1	The number of lines of code excluding comments
SIZE2	The total count of the number of data attributes and the number of local methods in a class

centage of changed instances. These three datasets have the least percentage of the changed instances. This will have on impact on the 'instance selection' step of our method, since the final model relies on learning from the selected instances. Therefore, the percentage of changed instances might be a threat of applying our method. However, considering the performance in average of 14 datasets, our unsupervised methods do not need prior labeled data but achieve competitive or even better performance than supervised within-project prediction methods. Hence, our method offers a viable solution for unlabeled dataset.

In addition, we also compute the weighted average performance in Table 3 based on the size of each dataset (i.e., the number of instances as Table 1 shows). In terms of CCR, our unsupervised methods CLAMI/CLAMI+ perform worse. The reason is that CLAMI/CLAMI+ perform worse in terms of CCR at some large datasets, such as 'argouml-0.32.2' and 'azureus-4.1.0.4'. These two datasets have the highest number of instances. In terms of AUC and F-measure, it shows that our conclusion remained no change, our unsupervised methods outperform the five baselines in average.

5.2. Answer of RQ2: performance of the unsupervised and cross-project method

The application scene of unsupervised and cross-project (CP) method are similar. Both of them are suitable for building prediction model for unlabeled datasets. In this section, we compare the unsupervised method with cross-project prediction method on the same target datasets.

In cross-project prediction, there is a source and target in each pair. The source is the training set and the target is the testing set. Concretely, we use the previous version in Table 1 as the source and the target version as the target. Fig. 4 shows the CCR, AUC and F-measure in different cross-project prediction pairs. For each target data, there are 13 ⟨source, target⟩ pairs (14 projects except for the one previous version in the same project). In Fig. 4, the x-axis represents the target data, the y-axis represents the source data, and the color (i.e., the gray degree) in each entry represents the performance. For example, when the target dataset is 'ant-1.8.2.0' (the first column), we will build 13 cross-project models by using 13 previous version data in other 13 projects (i.e., from '<antlr-3.3.0, ant-1.8.2.0>' to '<weka 3.5.7, ant-1.8.2.0>'). The case which adopts the 'ant-1.8.1.0' as the source is not considered in this RQ (we will use 0 to represent the performance in this RQ), since it is a within-project case which has been investigated in RQ1. From Fig. 4, we find that the performance is not stable due to different source. For example, when target on 'ant-1.8.2.0', the CCR is generally good in most pairs except for training on 'lucene-3.6.2.0' and 'weka-3.5.7'. When target on 'freemind-0.6.7', most of the source are not suitable for training. The reason behind the not stable performance is the difference of data distribution [17] between the source and target.

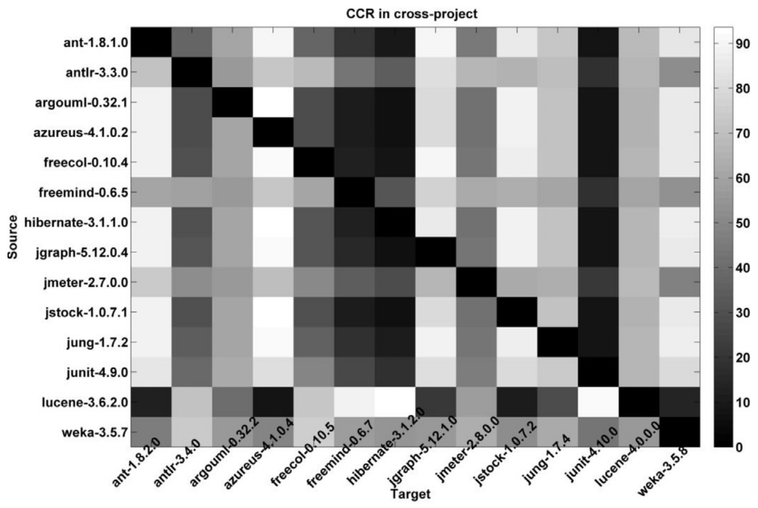
In order to compare the performance of the unsupervised and cross-project methods, we compute the average performance among 13 cross-project pairs for each target version. In detail, Table 4 lists the comparison among 14 projects. Overall, the unsupervised methods CLAMI and CLAMI+ show comparable performance to the cross-project prediction method. In particular, considering the CCR measure, the unsupervised methods outperform cross-project method among 7 datasets. In the dataset like ant-1.8.2.0, the unsupervised methods perform worse. However, considering the average of all datasets, unsupervised methods CLAMI and CLAMI+ improve them by 5.4% and 7.9% in terms of CCR. Considering the AUC measure, unsupervised methods CLAMI and CLAMI+ outperform cross-project method among 13 datasets and improve them by 14.5% and 18.2% in average of all datasets. Considering the F-measure, unsupervised methods CLAMI and CLAMI+ outperform cross-project method among 10 datasets and improve them by 35.1% and 40.5% in average of all datasets. Note that the advantage of unsupervised method is that they do not need prior labeled data from other projects but achieve comparable performance than cross-project prediction method.

5.3. Answer of RQ3: performance of the CLAMI and CLAMI+

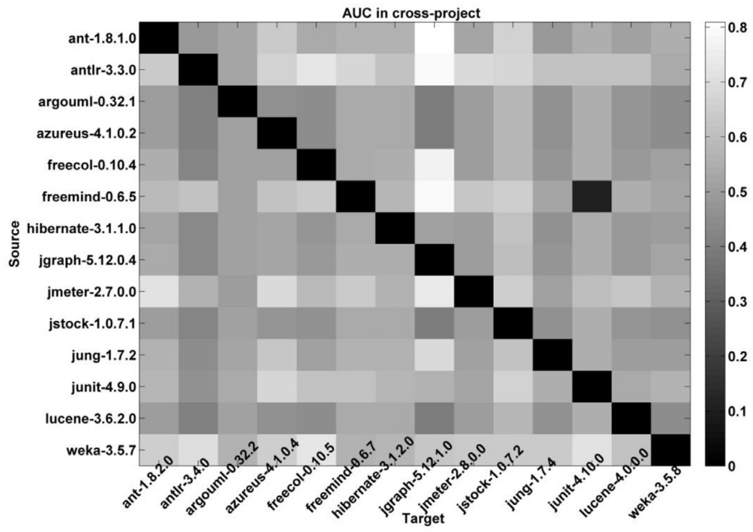
Table 3 shows that the performance of CLAMI+ achieves comparable or better result than CLAMI method. Only one out of the 14 datasets in which the CLAMI+ shows the worse result than CLAMI considering AUC, CCR or F-measure. In other cases, the CLAMI+ performs better or at least the same with CLAMI. Also, considering the average of all datasets, the CLAMI+ method improves the CLAMI method by 2.9% in CCR, 3.2% in AUC and 4% in F-measure.

In addition, we adopt the Friedman test to test whether the difference of the performance of all the methods (i.e., LB, MLP, RBF, SVM, SK, CP, CLAMI and CLAMI+) are statistically significant or not by following the guideline of Demšar [44]. The Friedman test results are as Table 5 shows. We list the average ranks (the approach with the best performance is ranked in "8") and the significant level *p-value*. In terms of CCR, the CLAMI and CLAMI+ show a comparable rank although not the best, and the CLAMI+ outperforms CLAMI. The p-value is greater than 0.05 which means the difference at CCR is not statistically significant between all the 8 methods. In terms of AUC and F-measure, the CLAMI and CLAMI+ show the higher ranks than other six baselines and the CLAMI+ is the best. And the *p-values* in both of the two measures are smaller than 0.05 which suggests that the difference between the average ranks of all the methods is statistically significant in terms of the two performance measures.

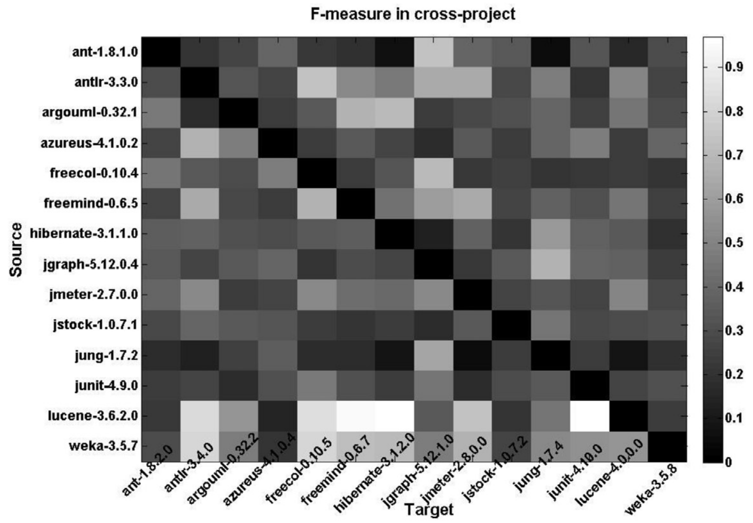
Subsequently, we conduct a post-hoc test (i.e., the Nemenyi test) for each pair of the methods by following the guidelines [17,44]. We use Fig. 5 to show the results of the Nemenyi test by following the visualization technique of Yan et al. [45]. Fig. 5(a) shows the results of CCR, Fig. 5(b) shows the results of AUC and Fig. 5(c) shows the results of F-measure. The x-axis value repre-



(a) CCR in cross-project predicton

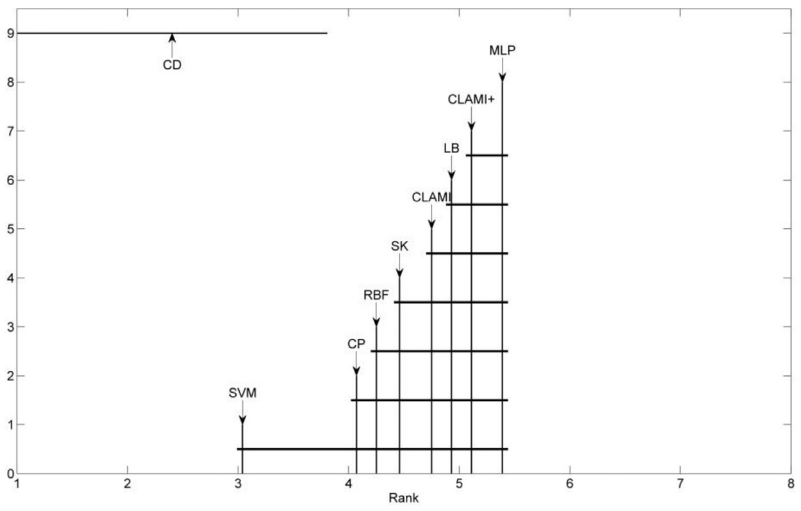


(b) AUC in cross-project predicton

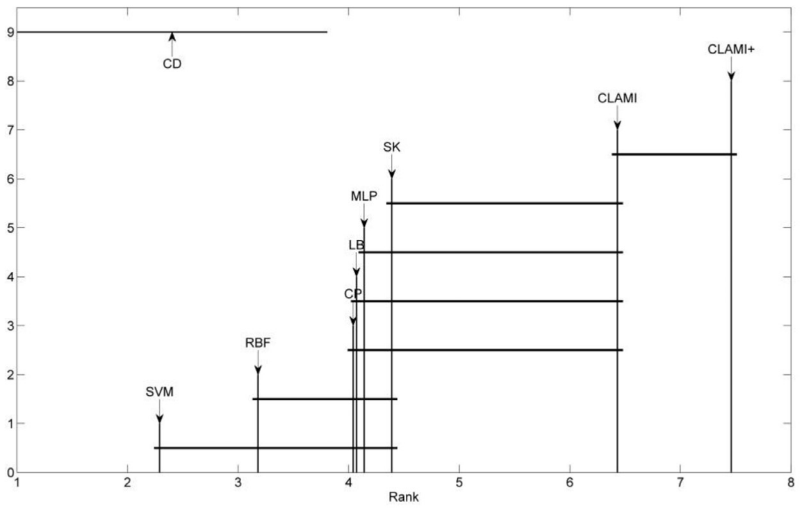


(c) F-measure in cross-project predicton

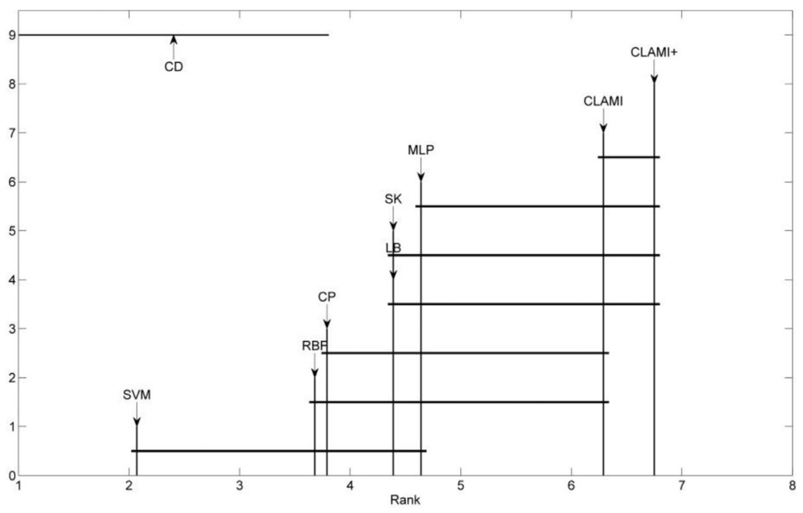
Fig. 4. CCR and AUC in cross-project predicton. Each square represents a prediction pair (source- > target). The diagonal squre is not considered in this RQ, since it has been investigated in RQ1.



(a) Nemenyi test of CCR



(b) Nemenyi test of AUC



(c) Nemenyi test of F-measure

Fig. 5. The average ranks comparison of all the baselines and unsupervised methods by using the Nemenyi test. Methods that are not significantly different are connected. Fig. 5(a) shows the results of CCR, Fig. 5(b) shows the results of AUC and Fig. 5(c) shows the results of F-measure.

Table 3
Performance comparison between unsupervised methods and five baselines. If the performance of the unsupervised methods CLAMI/CLAMI+ outperforms all the five baselines, the results are in bold. The better results between CLAMI and CLAMI+ are underlined.

(a) CCR Comparison

Project	CCR						
	LB	MLP	RBF	SVM	SK	CLAMI	CLAMI+
‘ant-1.8.2.0’	88.63	89.22	88.63	87.80	61.85	58.29	58.29
‘antlr-3.4.0’	56.85	53.53	46.47	36.51	37.76	**65.56**	**65.56**
‘argouml-0.32.2’	60.33	60.33	60.33	60.33	28.31	46.05	52.23
‘azureus-4.1.0.4’	92.32	92.38	92.29	92.29	91.17	52.83	52.83
‘freecol-0.10.5’	29.93	30.10	28.26	28.26	32.27	**63.88**	**64.05**
‘’freemind-0.6.7’	39.19	43.24	32.43	24.32	47.57	**52.70**	**60.81**
‘hibernate-3.1.2.0’	6.92	8.22	6.38	6.38	41.73	**50.70**	**57.62**
‘jgraph-5.12.1.0’	84.91	86.79	88.68	83.02	88.68	69.81	69.81
‘jmeter-2.8.0.0’	52.89	57.59	51.08	50.84	56.87	**63.98**	**66.63**
‘jstock-1.0.7.2’	89.49	89.49	89.13	88.41	70.65	55.07	55.07
‘jung-1.7.4’	72.65	72.01	71.58	71.15	69.66	52.35	51.50
‘junit-4.10.0’	12.88	14.72	9.20	7.98	26.99	**52.76**	**52.76**
‘lucene-4.0.0.0’	34.35	34.35	34.35	34.35	56.94	**65.97**	**65.97**
‘weka-3.5.8’	36.10	33.87	37.62	21.00	44.98	**55.59**	**55.67**
Average	54.10	54.70	52.60	49.47	53.96	**57.54**	**59.20**
Weighted average	61.49	61.77	61.00	58.84	59.50	55.03	56.71

(b) AUC Comparison

Project	AUC						
	LB	MLP	RBF	SVM	SK	CLAMI	CLAMI+
‘ant-1.8.2.0’	0.60	0.60	0.58	0.51	0.66	0.65	0.65
‘antlr-3.4.0’	0.63	0.60	0.55	0.47	0.48	**0.65**	**0.65**
‘argouml-0.32.2’	0.51	0.51	0.51	0.51	0.31	0.49	**0.52**
‘azureus-4.1.0.4’	0.47	0.47	0.47	0.47	0.58	**0.64**	**0.64**
‘freecol-0.10.5’	0.47	0.47	0.45	0.45	0.49	**0.67**	**0.67**
‘freemind-0.6.7’	0.66	0.62	0.56	0.58	0.50	0.61	0.63
‘hibernate-3.1.2.0’	0.54	0.55	0.54	0.54	0.48	**0.61**	**0.62**
‘jgraph-5.12.1.0’	0.65	0.67	0.73	0.49	0.65	0.72	**0.77**
‘jmeter-2.8.0.0’	0.56	0.60	0.56	0.58	0.61	**0.65**	**0.67**
‘jstock-1.0.7.2’	0.61	0.61	0.60	0.57	0.55	**0.62**	**0.66**
‘jung-1.7.4’	0.49	0.48	0.47	0.47	0.54	0.51	**0.56**
‘junit-4.10.0’	0.57	0.59	0.55	0.55	0.65	**0.76**	**0.76**
‘lucene-4.0.0.0’	0.47	0.47	0.47	0.47	0.54	**0.65**	**0.65**
‘weka-3.5.8’	0.52	0.48	0.53	0.45	0.60	0.59	0.58
Average	0.55	0.55	0.54	0.51	0.55	**0.63**	**0.65**
Weighted average	0.52	0.51	0.51	0.49	0.53	**0.61**	**0.62**

(c) F-measure Comparison

Project	F-measure						
	LB	MLP	RBF	SVM	SK	CLAMI	CLAMI+
‘ant-1.8.2.0’	0.30	0.31	0.26	0.25	0.31	0.30	0.30
‘antlr-3.4.0’	0.59	0.54	0.42	0.21	0.23	**0.72**	**0.72**
‘argouml-0.32.2’	0.48	0.45	0.42	0.44	0.28	0.46	0.43
‘azureus-4.1.0.4’	0.12	0.12	0.13	0.11	0.32	0.21	0.21
‘freecol-0.10.5’	0.44	0.45	0.60	0.43	0.11	**0.69**	**0.70**
‘freemind-0.6.7’	0.48	0.54	0.40	0.26	0.62	**0.65**	**0.73**
‘hibernate-3.1.2.0’	0.52	0.53	0.60	0.53	0.70	0.65	**0.72**
‘jgraph-5.12.1.0’	0.50	0.53	0.63	0.31	0.63	0.56	0.56
‘jmeter-2.8.0.0’	0.45	0.55	0.39	0.31	0.48	**0.65**	**0.70**
‘jstock-1.0.7.2’	0.17	0.17	0.12	0.13	0.18	**0.24**	**0.24**
‘jung-1.7.4’	0.42	0.41	0.39	0.37	0.32	0.39	**0.47**
‘junit-4.10.0’	0.52	0.44	0.43	0.42	0.34	**0.65**	**0.65**
‘lucene-4.0.0.0’	0.51	0.51	0.51	0.51	0.43	**0.58**	**0.58**
‘weka-3.5.8’	0.26	0.24	0.26	0.23	0.34	0.30	0.30
Average	0.41	0.41	0.40	0.32	0.38	**0.50**	**0.52**
Weighted average	0.33	0.33	0.33	0.29	0.35	**0.41**	**0.42**

sents the average rank of each method. The y-axis which corresponds to an arrow represents the index of a method (8 methods in total). In addition, the 9th index at y-axis represents the critical difference (referred to as CD), its value is represented by the line's length. In detail, it is a critical value, namely, the average ranks of the two methods is significantly different when the difference of the average ranks exceeds CD value [44]. The CD is impacted by the significance level α ($\alpha = 0.05$ in this case), the number of compared methods k ($k = 8$ in this case) and the number of the datasets N ($N = 14$ in this case). It is computed as Formula

Table 4
Performance comparison between unsupervised methods and cross-project (CP) prediction method. If the performance of the unsupervised methods CLAMI/CLAMI+ outperforms the cross-project method, the results are in bold.

Project	CCR			AUC			F-measure		
	CP	CLAMI	CLAMI+	CP	CLAMI	CLAMI+	CP	CLAMI	CLAMI+
'ant-1.8.2.0'	74.21	58.29	58.29	0.56	**0.65**	**0.65**	0.31	0.30	0.30
'antlr-3.4.0'	41.94	**65.56**	**65.56**	0.48	**0.65**	**0.65**	0.43	**0.72**	**0.72**
'argouml-0.32.2'	57.71	46.05	52.23	0.52	0.49	**0.52**	0.34	**0.46**	**0.43**
'azureus-4.1.0.4'	76.49	52.83	52.83	0.58	**0.64**	**0.64**	0.30	0.21	0.21
'freecol-0.10.5'	45.95	**63.88**	**64.05**	0.55	**0.67**	**0.67**	0.44	**0.69**	**0.70**
'freemind-0.6.7'	27.86	**52.70**	**60.81**	0.57	**0.61**	**0.63**	0.41	**0.65**	**0.73**
'hibernate-3.1.2.0'	24.24	**50.70**	**57.62**	0.56	**0.61**	**0.62**	0.40	**0.65**	**0.72**
'jgraph-5.12.1.0'	75.04	69.81	69.81	0.61	**0.72**	**0.77**	0.45	**0.56**	**0.56**
'jmeter-2.8.0.0'	49.04	**63.98**	**66.63**	0.54	**0.65**	**0.67**	0.39	**0.65**	**0.70**
'jstock-1.0.7.2'	72.88	55.07	55.07	0.62	**0.62**	**0.66**	0.27	0.24	0.24
'jung-1.7.4'	65.96	52.35	51.50	0.51	**0.51**	**0.56**	0.40	0.39	**0.47**
'junit-4.10.0'	19.77	**52.76**	**52.76**	0.53	**0.76**	**0.76**	0.37	**0.65**	**0.65**
'lucene-4.0.0.0'	65.21	**65.97**	**65.97**	0.53	**0.65**	**0.65**	0.34	**0.58**	**0.58**
'weka-3.5.8'	71.68	55.59	55.67	0.51	**0.59**	**0.58**	0.27	**0.30**	**0.30**
Average	54.86	**57.54**	**59.20**	0.55	**0.63**	**0.65**	0.37	**0.50**	**0.52**

Table 5
Friedman test for the performance comparison.

Measure	Average rank								*p-value*
	LB	MLP	RBF	SVM	SK	CP	CLAMI	CLAMI+	
CCR	4.93	5.39	4.25	3.04	4.46	4.07	4.75	5.11	0.243
AUC	4.07	4.14	3.18	2.29	4.39	4.04	6.43	7.46	0.000
F-measure	4.39	4.64	3.68	2.07	4.39	3.79	6.29	6.75	0.000

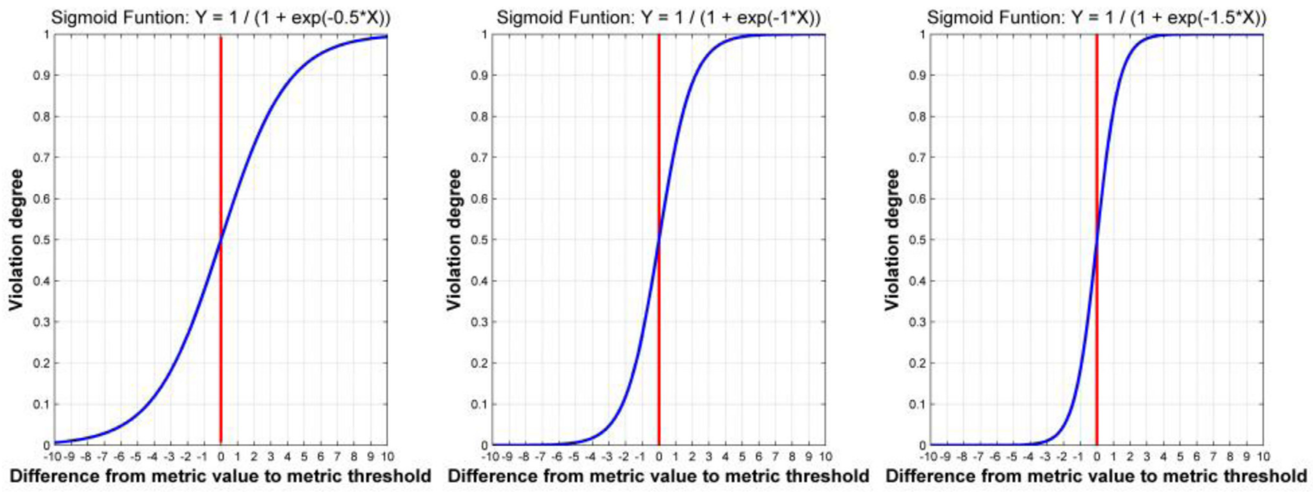


Fig. 6. Illustration of the difference in the curve of sigmoid function by varying the steepness.

(3) shows [44]. The q_α is decided by referring the value table suggested by Demšar [44] which is related with the significance level and the number of compared methods ($q_\alpha = 3.031$ in our case).

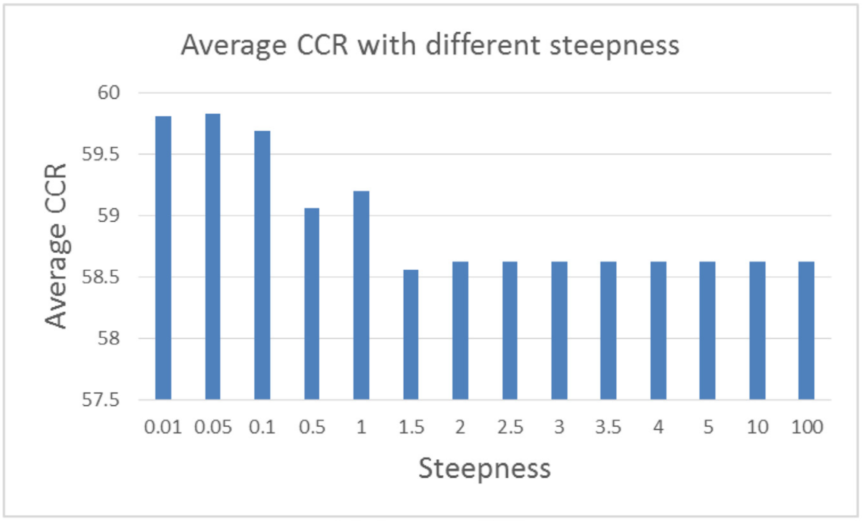
$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{3}$$

The connected line between two methods in Fig. 5 represents that the difference of the average ranks between the two methods is less than CD. It represents that there is not a statistically significant difference between the two methods. From Fig. 5(a), we observe that the differences of the average ranks of all the pairs are not statistically significant in terms of CCR, since all the methods are connected. From Fig. 5(b), we observe that CLAMI+ outperforms the SVM, RBF, CP, MLP, SK and LB with statistical significance in terms of AUC. In addition, CLAMI outperforms the SVM and RBF with statistical significance. From Fig. 5(c), we observe that CLAMI+ outperforms the SVM, RBF and CP with statistical significance. And the average ranks between CLAMI and CLAMI+ do not show a significant difference in AUC and F-measure.

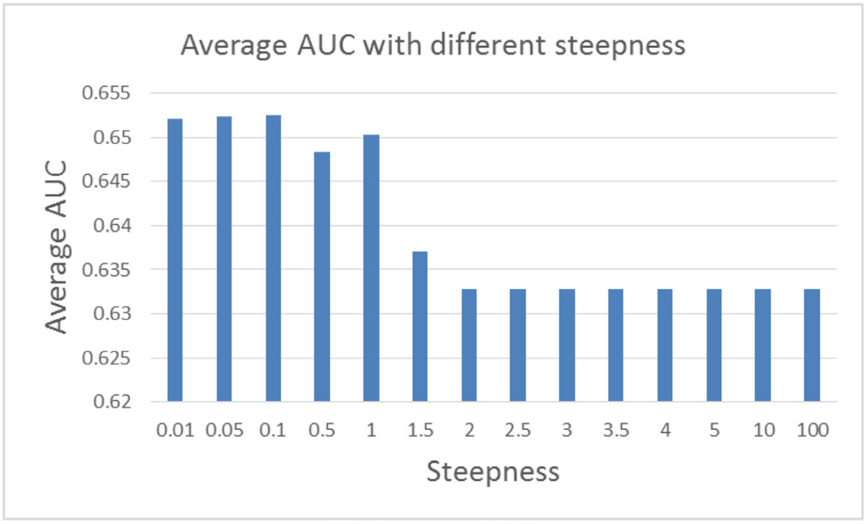
Overall, In terms of CCR, the unsupervised methods show a comparable rank although not the best. In terms of AUC and F-measure, the unsupervised methods show the higher ranks than all the baselines. In addition, CLAMI+ shows the better ranks than CLAMI in both CCR, AUC and F-measure although they are not statistically significant.

5.4. Answer of RQ4: effect of varying the steepness of sigmoid function

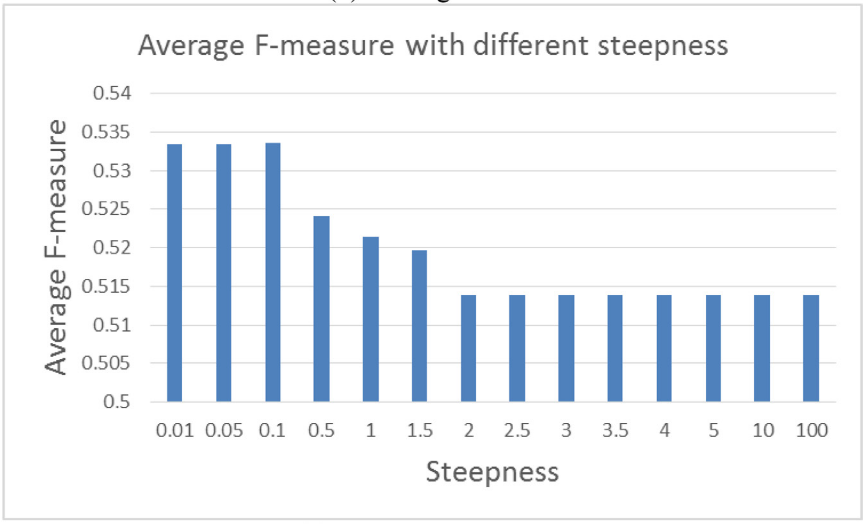
In this work, we adopt the sigmoid function to transform the difference between metric value and metric threshold to the violation degree. The sensitivity of the transformation with the difference is impacted by the steepness of the function. Fig. 6 illustrates the different curves of the function by setting different steepness



(a) Average CCR



(b) Average AUC



(c) Average F-measure

Fig. 7. The average performance by varying the steepness of sigmoid function.

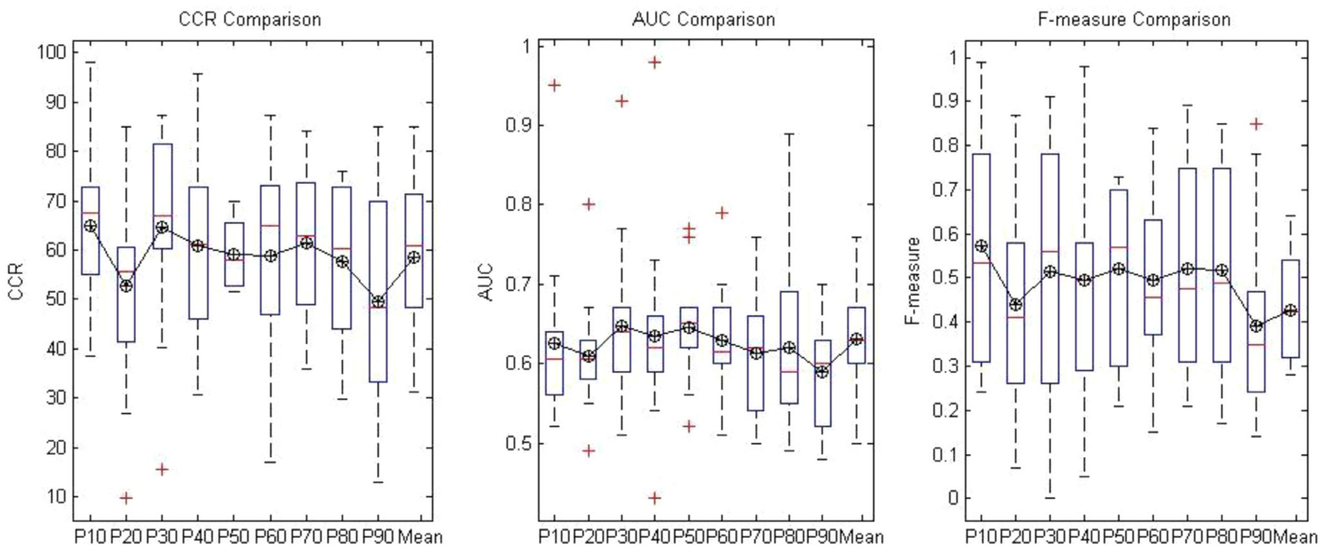


Fig. 8. Distribution comparison of the performance by varying the threshold. Each box represents the performance distribution of all the datasets at a threshold. The line plot represents the average performance.

from *S* = *0.5* to *S* = *1.5* (stepped by *0.5*). It shows that if the steepness value is greater, the function curve is steeper, the working range in the x-axis (e.g., *[−6, 6]* when *S* = *1*) is more narrow. The working range corresponds to the results of transformation which can further impact our clustering and labeling step.

In order to explore the effect of steepness to our performance, we conduct the experiments by varying steepness in a certain range. Fig. 7 shows the results of performance by varying the steepness. Overall, our finding suggests that: (1) the performance is slightly impacted by the steepness. Concretely, the average CCR ranges from 58.618 to 59.828, the average AUC ranges from 0.633 to 0.652 and the average F-measure ranges from 0.514 to 0.534. (2) The steepness smaller than 1 has better performance than the one greater than 1. The reason is that the transformation of the function is slower when smaller than 1. As a result, the output is more diverse which records more informative values of the difference between metric values and metric threshold. However, it is hasty to state that the smaller the better, since it does not follow the pattern certainly. For example, we manually check the results of each project. The best steepness choice of 'ant-1.8.2' is 1 while the best steepness choice of 'antlr-3.4.0' is 0.05. Therefore, generally, the performance is better when the steepness is smaller than 1 and the best choice of the steepness is different which depends on the datasets. (3) The performance is stable when the steepness is greater than a certain value. From Fig. 7, we found that the performance keep fixed when the steepness is greater than 2. This indicates that the performance converges when the transformation does not impact the following step of our method (i.e., labeling). The reason is that when the steepness is greater than a certain value, the working range of x-axis is smaller than a threshold. Meanwhile, the output of the K value (in our clustering step) does not impact the clustering result. As a result, the clustering results are the same, and the labeling results are also the same in those cases. Therefore, the models and the performances are the same when the steepness is greater than a certain value.

5.5. Answer of RQ5: effect of varying the metric threshold

To investigate the effect of the threshold, we adopt various cutoff metric values as the threshold: *n-th* percentiles where n is 10, 20, …, 90 and the mean value. Totally, we use 10 thresholds, namely, P10 (i.e., the 10th percentile), P20, P30, P40, P50 (i.e., median), P60, P70, P80, P90 and Mean. Fig. 8 and Table 6 provide the experimental results. In detail, Fig. 8 shows the performance distribution comparison at different thresholds. Each box represents the performance distribution at a certain threshold. In addition, in order to provide the overall insight, we use the line plot to represent the mean of the performance of all the datasets at different thresholds. Table 6 provides the detailed performance values of all the datasets at different thresholds.

In summary, our finding suggests that: (1) the choice of the threshold has an effect to the performance of our method. Through the boxplot, we found that there is a significant difference at different thresholds, including the median, mean and the degree of dispersion of each box. Through Table 6, we found that there is a significant difference for a certain dataset when choosing different threshold. For example, the best CCR for 'ant-1.8.2.0' is 87.2 at P30 and the worst is 12.91 at P90. (2) The threshold P50 (i.e., median) is not the best choice considering the average performance for all the datasets. In detail, the best choice is P10 considering the average CCR and F-measure. The best choice is P30 and P50 considering the average AUC. Besides, through the spacing of the box in Fig. 8 and the standard deviation value in Table 6, we observe that the degree of dispersion is smaller when setting threshold as P50 considering both CCR and AUC. This suggests that the median can be regarded as an empirical choice when we have no prior knowledge for deciding better threshold. (3) The best choice of the threshold depends on the dataset. In detail, through Table 6, we observe that the best choice for different datasets are various. The reason comes from the problem of using thresholds. In general, it is stated that a threshold value woks well in one setting must not necessarily be good every setting [46]. It depends on various factors which are project dependent, such as the organization, the tools used and the qualification of the developers. Therefore, a more specific thresholds deciding method for different project is required to enhance our method.

6. Threats to validity

6.1. Internal validity

Threats to internal validity result from the potential limitations in our experiments.

Table 6
The performance of different datasets at different thresholds. The best choice for each dataset are in bold.

Threshold Dataset	P10	P20	P30	P40	P50	P60	P70	P80	P90	Mean
(a) CCR with different thresholds										
‘ant-1.8.2.0’	38.51	35.55	**87.20**	71.45	58.29	86.02	67.18	72.75	12.91	71.45
‘antlr-3.4.0’	71.37	80.50	64.32	30.71	65.56	47.30	48.96	**75.93**	74.27	56.02
‘argouml-0.32.2’	57.08	55.28	**60.40**	56.74	52.23	47.04	57.61	46.05	59.14	54.68
‘azureus-4.1.0.4’	65.66	**85.05**	52.51	39.49	52.83	16.89	84.25	29.75	63.71	73.84
‘freecol-0.10.5’	74.25	45.32	60.20	**80.77**	64.05	72.91	73.58	40.80	37.79	48.33
‘freemind-0.6.7’	50.00	58.11	**83.78**	45.95	60.81	32.43	81.08	66.22	21.62	35.14
‘hibernate-3.1.2.0’	65.62	9.62	65.62	37.51	57.62	72.97	**80.32**	58.16	28.97	31.24
‘jgraph-5.12.1.0’	69.50	41.51	**86.79**	54.72	69.81	77.36	39.62	62.26	84.91	84.91
‘jmeter-2.8.0.0’	**73.37**	60.48	69.40	66.87	66.63	66.63	70.00	70.00	72.41	60.36
‘jstock-1.0.7.2’	55.07	26.81	15.58	72.83	55.07	**87.32**	42.03	34.06	33.33	72.83
‘jung-1.7.4’	48.93	**69.02**	68.59	51.50	51.50	56.62	58.55	58.55	37.18	61.54
‘junit-4.10.0’	**98.16**	60.12	81.60	95.71	52.76	31.90	52.76	75.46	37.42	37.42
‘lucene-4.0.0.0’	69.52	56.13	68.87	65.65	65.97	63.55	67.74	44.03	**70.00**	67.74
‘weka-3.5.8’	72.83	54.07	40.30	**83.29**	55.67	66.22	35.75	75.34	59.96	65.06
Average	64.99	52.68	64.65	60.94	59.20	58.94	61.39	57.81	49.55	58.61
Standard deviation	14.00	19.37	19.40	19.02	6.22	21.28	15.99	16.12	22.27	15.94
(b) AUC with different thresholds										
‘ant-1.8.2.0’	0.63	0.61	0.64	0.67	0.65	0.66	0.65	**0.72**	0.50	0.67
‘antlr-3.4.0’	0.59	**0.67**	0.65	0.43	0.65	0.56	0.56	0.56	0.48	0.62
‘argouml-0.32.2’	0.57	**0.59**	0.51	0.59	0.52	0.56	0.51	0.49	0.52	0.50
‘azureus-4.1.0.4’	0.54	0.64	0.65	0.59	0.64	0.51	0.65	0.56	0.67	**0.68**
‘freecol-0.10.5’	0.52	0.58	0.67	**0.73**	0.67	0.70	0.50	0.54	0.53	0.60
‘freemind-0.6.7’	0.56	0.62	0.70	0.59	0.63	0.63	**0.71**	0.66	0.57	0.65
‘hibernate-3.1.2.0’	0.62	0.56	0.62	**0.63**	0.62	0.60	0.56	0.62	0.62	0.63
‘jgraph-5.12.1.0’	0.66	0.55	0.77	0.65	0.77	**0.79**	0.54	0.70	0.62	0.76
‘jmeter-2.8.0.0’	**0.71**	0.63	0.64	0.61	0.67	0.67	0.69	0.69	0.70	0.63
‘jstock-1.0.7.2’	0.62	0.60	0.59	**0.66**	0.66	0.60	0.66	0.64	0.63	0.66
‘jung-1.7.4’	0.58	0.49	0.51	0.56	0.56	0.60	**0.61**	0.51	0.52	0.52
‘junit-4.10.0’	0.95	0.80	0.93	**0.98**	0.76	0.67	0.76	0.89	0.69	0.69
‘lucene-4.0.0.0’	0.64	0.61	0.62	0.65	0.65	**0.66**	0.63	0.55	0.61	0.63
‘weka-3.5.8’	0.56	0.58	0.57	0.54	0.58	**0.60**	0.54	0.56	0.59	0.59
Average	0.63	0.61	0.65	0.64	0.65	0.63	0.61	0.62	0.59	0.63
Standard deviation	0.11	0.07	0.11	0.12	0.07	0.07	0.08	0.11	0.07	0.07
(c) F-measure with different thresholds										
‘ant-1.8.2.0’	0.27	0.26	0.36	0.35	0.30	0.37	0.32	**0.39**	0.22	0.35
‘antlr-3.4.0’	0.81	**0.87**	0.70	0.05	0.72	0.42	0.45	0.85	0.85	0.57
‘argouml-0.32.2’	0.50	0.56	0.00	0.55	0.43	**0.59**	0.21	0.46	0.14	0.31
‘azureus-4.1.0.4’	**0.31**	0.31	0.21	0.18	0.21	0.15	0.31	0.17	0.24	0.28
‘freecol-0.10.5’	0.85	0.40	0.63	**0.87**	0.70	0.80	0.84	0.31	0.25	0.45
‘freemind-0.6.7’	0.63	0.71	**0.91**	0.58	0.73	0.39	0.89	0.78	0.22	0.43
‘hibernate-3.1.2.0’	0.78	0.07	0.78	0.51	0.72	0.84	**0.89**	0.72	0.39	0.42
‘jgraph-5.12.1.0’	0.54	0.42	**0.70**	0.48	0.56	0.63	0.41	0.52	0.43	0.64
‘jmeter-2.8.0.0’	0.78	0.58	**0.79**	0.78	0.70	0.70	0.75	0.75	0.78	0.56
‘jstock-1.0.7.2’	0.24	0.23	0.22	**0.29**	0.24	0.22	0.26	0.24	0.24	0.29
‘jung-1.7.4’	0.49	0.19	0.26	0.47	0.47	0.49	**0.50**	0.36	0.47	0.35
‘junit-4.10.0’	**0.99**	0.72	0.89	0.98	0.65	0.41	0.65	0.85	0.48	0.48
‘lucene-4.0.0.0’	0.53	0.56	0.49	0.57	0.58	**0.59**	0.54	0.53	0.46	0.54
‘weka-3.5.8’	0.30	0.29	0.28	0.29	0.30	**0.32**	0.27	0.30	0.31	0.32
Average	0.57	0.44	0.52	0.50	0.52	0.49	0.52	0.52	0.39	0.43
Standard deviation	0.24	0.23	0.29	0.26	0.19	0.21	0.24	0.23	0.21	0.12

Impact of the threshold. The clustering and labeling phase is impacted by the choice of metric threshold. There are various methods for deciding a metric threshold. The results show that the choice of threshold depends on the dataset. However, we did not provide the threshold deciding method. This might be a threat to our work. In this work, we adopt a typical threshold (i.e., the median) and explore the effect of different thresholds to mitigate this issue. A more refined work is to propose a method of automatically deciding the threshold for each dataset.

Impact of sigmoid function. The difference of our proposed CLAMI+ method is that we transform the 1 or 0 result (violation or not) to a continuous value ranging from 0 to 1 which represents the violation degree by using the sigmoid function. The parameter of sigmoid function, such as steepness, has an impact on the transformation. However, we adopt the regular sigmoid function on all the metrics. This might be a limitation to the performance of CLAMI+, since different metrics possess different distribution and they may suitable for different parameter settings. We explore the impact of different settings of steepness to mitigate this problem. A more refined work is to investigate the different settings at different metrics.

6.2. External validity

Threats to external validity correlate with the generalizability of our approach.

Impact of project features. The proposed methods are validated on the adopted versions of open-source projects. The generalization of the results may be an external threat. Different projects have different properties, such as programming languages, duration between releases and software maturity. These properties are confounding factors which may impact the results. Thus, how to select the experimental projects may be a threat to the validity. However, we adopted 14 projects to mitigate this threat. In terms of the programming language, since the adopted datasets are all Java systems, the method may not work well when generalizing to other languages. The reason is that different programming languages may have different code metrics. For example, the object-oriented (OO) metrics are only existed in OO languages. However, If the generalized language has the similar metric rationale (i.e., the change-prone tendency), the problem can be mitigated.

Impact of the project version features. There are multiple release versions in a software lifecycle. Different software versions have different features, such as types and percentage of changes. Currently, we does not draw a conclusion on how does it impact our method. Thus, how to select the experimental versions may be a threat to the validity. However, we adopted 14 projects with the same version selection manner to mitigate this threat. To eliminate this threat further, a more refined work is needed to investigate the proposed method on massive versions in a variety of projects.

7. Conclusion and future work

In this paper, we proposed to adopt unsupervised approach to tackle change-prone class prediction on unlabeled datasets. Concretely, we applied a state-of-art unsupervised method CLAMI and proposed a novel approach CLAMI+ by extending CLAMI. The unsupervised approach was evaluated on 14 open source projects. In addition, we compared its performance against typical within-project prediction methods (learning from historical labeled data within a project) and cross-project prediction methods (learning from another labeled project). The results showed that the unsupervised methods yield comparable or better results to the baselines.

In summary, we drew the conclusions as follows: first, compared with the within-project prediction methods in average, CLAMI and CLAMI+ improved them by 5.2%−19.7% in terms of CCR, 13.9%−27.2% in terms of AUC and 21.7%−61.9% in terms of F-measure. Second, compared with the cross-project prediction method, the unsupervised methods CLAMI and CLAMI+ improved it by 5.4% and 7.9% in terms of CCR, by 14.5% and 18.2% in terms of AUC, by 35.1% and 40.5% in terms of F-measure. Third, the performance of CLAMI+ outperforms the CLAMI method. Considering the average of all datasets, the CLAMI+ method improves the CLAMI method by 2.9% in CCR, 3.2% in AUC and 4% in F-measure. Fourth, the steepness of the sigmoid function and the choice of threshold are two factors which have an impact on the proposed method.

In the future, we plan to enhance the effectiveness of our approach further from three aspects. First, we plan to add some standard machine learning clustering methods (e.g., k-means or fuzzy c-means) in the grouping step. Second, we plan to investigate the impact of data imbalance and propose to use imbalance-aware technique to enhance our method. Third, we plan to improve the performance by proposing an adaptive method to determine the optimized parameters including the threshold and the setting of sigmoid function.

Acknowledgments

The work described in this paper was partially supported by the National Natural Science Foundation of China (Grant no. 61402062 and 61602068), the Fundamental Research Funds for the Central Universities of China (Grant No. 106112017CDJXY090001) and Chongqing Research Program of Basic Science & Frontier Technology (No. cstc2017jcyjB0305).

References

[1] Y. Zhou, H. Leung, Predicting object-oriented software maintainability using multivariate adaptive regression splines, J. Syst. Softw. 80 (2007) 1349–1361.
[2] R. Malhotra, M. Khanna, Examining the effectiveness of machine learning algorithms for prediction of change prone classes, in: Proceedings of the nternational Conference on High Performance Computing & Simulation, 2014, pp. 635–642.
[3] M. Yan, Y. Fu, X. Zhang, D. Yang, L. Xu, J.D. Kymer, Automatically classifying software changes via discriminative topic model: supporting multi-category and cross-project, J. Syst. Softw. 113 (2016) 296–308.
[4] Y. Fu, M. Yan, X. Zhang, L. Xu, D. Yang, J.D. Kymer, Automated classification of software change messages by semi-supervised latent dirichlet allocation, Inf. Softw. Technol. 57 (2015) 369–377.
[5] R. Malhotra, M. Khanna, An exploratory study for software change prediction in object-oriented systems using hybridized techniques, Autom. Softw. Eng. (2016) 1–45.
[6] M. Elish, H. Aljamaan, I. Ahmad, Three empirical studies on predicting software maintainability using ensemble methods, Soft Comput. 19 (2015) 2511–2524.
[7] Z. Yuming, H. Leung, X. Baowen, Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness, IEEE Trans. Softw. Eng. 35 (2009) 607–623.
[8] F. Khomh, M. Di Penta, X Gue, X He, Y. Neuc, An exploratory study of the impact of code smells on software change-proneness, in: Proceedings of the 16th Working Conference on Reverse Engineering (WCRE 2009), 2009, pp. 75–84.
[9] D. Posnett, C. Bird, P. Dévanbu, An empirical study on the influence of pattern roles on change-proneness, Empir. Softw. Eng. 16 (2011) 396–423.
[10] M.O. Elish, M. Al-Rahman Al-Khiaty, A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software, J. Softw. 25 (2013) 407–437.
[11] S. Eski, F. Buzluca, An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes, in: Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011, pp. 566–571.
[12] C. van Koten, A.R. Gray, An application of Bayesian network for predicting object-oriented software maintainability, Inf. Softw. Technol. 48 (2006) 59–67.
[13] M. Amoui, M. Salehie, L. Tahvildari, Teporal software change prediction using neural networks, Int. J. Softw. Eng. Knowl. Eng. 19 (2009) 995–1014.
[14] R. Malhotra, A.J. Bansal, Cross project change prediction using open source projects, in: Proceedings of the International Conference on Advances in Computing, Communications and Informatics, 2014, pp. 201–207.
[15] J. Nam, S.J. Pan, S. Kim, Transfer defect learning, in: Proceedings of the International Conference on Software Engineering, „ IEEE Press, San Francisco, CAUSA, 2013, pp. 382–391.
[16] A. Panichella, R. Oliveto, A. De Lucia, Cross-project defect prediction models: L'Union fait la force, in: Proceedings of the IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering, 2014, pp. 164–173.
[17] J. Nam, S. Kim, CLAMI: defect prediction on unlabeled datasets, in: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, 2015, pp. 452–463.
[18] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (2010) 1345–1359.
[19] B. Turhan, On the dataset shift problem in software engineering prediction models, Empir. Softw. Eng. 17 (2012) 62–74.
[20] S. Zhong, T.M. Khoshgoftaar, N. Seliya, Unsupervised learning for expert-based software quality estimation, in: HASE, Citeseer, 2004, pp. 149–155.
[21] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, H. Leung, Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, Seattle, WA, USA, 2016, pp. 157–168.
[22] E. Tempero, C. Anslow, J. Dietrich, T. Han, L. Jing, M. Lumpe, H. Melton, J. Noble, TheQualitas Corpus, A curated collection of java code for empirical studies, in: Proceedings of the 17th Asia Pacific Software Engineering Conference, 2010, pp. 336–345.
[23] M. Yan, M. Yang, C. Liu, X. Zhang, Self-learning change-prone class prediction, in: Proceedings of the 28th International Conference on Software Engineering and Knowledge Engineering, 2016.
[24] E. Giger, M. Pinzger, H.C. Gall, Can we predict types of code changes? An empirical analysis, in: Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on, 2012, pp. 217–226.
[25] D. Godara, R. Singh, A new hybrid model for predicting change prone class in object oriented software, Int. J. Comput. Sci. Telecommun. 5 (2014) 1–6.
[26] A. Güneş Koru, H. Liu, Identifying and characterizing change-prone classes in two large-scale open-source products, J. Syst. Softw. 80 (2007) 63–73.
[27] H. Lu, Y. Zhou, B. Xu, H. Leung, L. Chen, The ability of object-oriented metrics to predict change-proneness: a meta-analysis, Empir. Softw. Eng. 17 (2012) 200–242.
[28] L.C. Briand, W.L. Melo, J. Wust, Assessing the applicability of fault-proneness models across object-oriented software projects, IEEE Trans. Softw. Eng. 28 (2002) 706–720.
[29] Z. He, F. Shu, Y. Yang, M. Li, Q. Wang, An investigation on the feasibility of cross-project defect prediction, Autom. Softw. Eng. 19 (2012) 167–199.
[30] F. Peters, T. Menzies, A. Marcus, Better cross company defect prediction, in: Proceedings of the 10th IEEE Working Conference on Mining Software Repositories, 2013, pp. 409–418.

[31] 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: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ACM, Amsterdam, The Netherlands, 2009, pp. 91–100.

[32] 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 Software Engineering 14 (2009) 540–578.

[33] R. Malhotra, M. Khanna, Mining the impact of object oriented metrics for change prediction using machine learning and search-based techniques, in: Proceedings of the International Conference on Advances in Computing, Communications and Informatics, 2015, pp. 228–234.

[34] R. Malhotra, A. Bansal, Prediction of change prone classes using threshold methodology, Advances in Computer Science and Information Technology vol. 2 (2015) 30–35.

[35] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (2007) 2–13.

[36] A.G. Koru, J. Tian, Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products, IEEE Trans. Softw. Eng. 31 (2005) 625–642.

[37] E. Kocaguneli, T. Menzies, J. Keung, D. Cok, R. Madachy, Active learning and effort estimation: Finding the essential content of software effort estimation data, IEEE Trans. Softw. Eng. 39 (2013) 1040–1053.

[38] Y.F. Li, M. Xie, T.N. Goh, A study of project selection and feature weighting for analogy based software cost estimation, J. Syst. Softw. 82 (2009) 241–252.

[39] M.O. Elish, K.O. Elish, Application of treenet in predicting object-oriented software maintainability: a comparative study, in: Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR 2009), 2009, pp. 69–78.

[40] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Trans. Softw. Eng. 20 (1994) 476–493.

[41] W. Li, S. Henry, Object-oriented metrics that predict maintainability, J. Syst. Softw. 23 (1993) 111–122.

[42] J.A. Hartigan, M.A. Wong, Algorithm AS 136: A k-means clustering algorithm, J. R. Stat. Soc. Ser. C 28 (1979) 100–108.

[43] F. Zhang, Q. Zheng, Y. Zou, A.E. Hassan, Cross-project defect prediction using a connectivity-based unsupervised classifier, in: Proceedings of the 38th International Conference on Software Engineering, ACM, Austin, Texas, 2016, pp. 309–320.

[44] J. Demsar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.

[45] M. Yan, X. Zhang, D. Yang, L. Xu, J.D. Kymer, A component recommender for bug reports using discriminative probability latent semantic analysis, Inf. Softw. Technol. 73 (2016) 37–51.

[46] S. Herbold, J. Grabowski, S. Waack, Calculation and optimization of thresholds for sets of software metrics, Empir. Softw. Eng. 16 (2011) 812–841.