Self-learning Change-prone Class Prediction

Meng Yan, Mengning Yang, Chao Liu, Xiaohong Zhang
School of Software Engineering
Chongqing University
Chongqing 401331, China
{meng.yan, mnyang, liu.chao, xhongz} @cqu.edu.cn

***Abstract*—Software change-prone class prediction can enhance software decision making activities during software maintenance (e.g., resource allocating). Many change-prone class prediction approaches have been proposed and most are effective in inter-version prediction within a project. These approaches usually build a supervised prediction model by learning from historical labeled dataset. However, a major challenge which remains is that this typical change-prone prediction setting cannot be used for new projects or projects with limited historical data. To address this challenge, we propose to tackle this task by adopting a novel prediction method which has not been used in change-prone prediction, namely self-learning method. The key idea of the self-learning method is to enable the change-prone prediction on new projects or projects with limited historical dataset by learning from itself. In this paper, we apply a state-of-art self-learning method, CLAMI, to change-prone prediction. In addition, we propose a novel self-learning approach CLAMI+ by extending CLAMI. The experiments among 14 open source projects show that the self-learning methods achieve comparable results to four typical inter-version baselines and the proposed CLAMI+ slightly improves the CLAMI method on average.**

Keywords-software maintenance; change-prone prediction; self-learning; empirical software engineering

I. INTRODUCTION

Software maintenance has been regarded as one of the most expensive and tough tasks in the whole software lifecycle [1]. Managing and controlling changes 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]. 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 [5]. 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 have been proved to correspond to the change-proneness, such as OO metrics (e.g., cohesion, coupling, inheritance, etc.) [6], code smells [7], design patterns and [8] evolution metrics [9, 10]. In terms of the techniques, different machine learning approaches have been used, such as Bayesian networks [11], neural networks [12], multivariate regression [1] and ensemble methods [5]. A typical prediction model based on machine learning is designed by learning from a historical labeled dataset in a supervised way. However, this technique is difficult to apply on new projects or projects with limited historical data.

A cross project change-prone class prediction method has been proposed to address the above-mentioned issue [13]. The cross project technique is motivated by the similar techniques in defect prediction [14, 15]. It enables change-prone class prediction on projects with limited labeled dataset by learning from other projects. Unfortunately, one issue which remains in cross-project prediction is that different datasets possess different distributions [16]. The success rate (ratio of combination whose performance is greater than a certain threshold) of cross-project reported in the work [13] is generally poor (30%) which cannot compare to the prediction performance (67%) of the methods using historical datasets (i.e., inter-version prediction within a project). This implies that the cross-project change-prone prediction may not be effective and it depends on the quality of the source project [13].

To address this issue, we propose to tackle this task by adopting self-learning method. In detail, we apply a state-of-art self-learning method (CLAMI: Clustering, LAbeling, Metric selection and Instance selection) to the change-prone class prediction which has been successfully used in defect prediction [16]. The key idea of this self-learning method is to enable the prediction on new projects or projects with limited labeled datasets by learning from itself.

The process of this self-learning method can be interpreted by dividing three phases as Figure 1 shows. The clue of the process is to build the prediction model by learning on selected informative metrics and instances of itself. In detail, the first phase is clustering and initialized labeling. In this phase, an unlabeled dataset is clustered and labeled according to the magnitude of metric values [16]. The motivation of this phase is to provide the initialized labels of all the instances. However, the initialized labels of all the instances might not be correct enough. In our self-learning method, some 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

DOI reference number: 10.18293/SEKE2016-039

modeling and prediction. The prediction model is built by learning from the selected instances in the second phase.

In particular, the initialized labeling step in the first phase of CLAMI method is conducted by measuring the count of violation (i.e., a metric value is greater than a certain threshold) of an instance. 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 self-learning method CLAMI+ by extending CLAMI. The difference lies in the first phase as Figure 1 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.

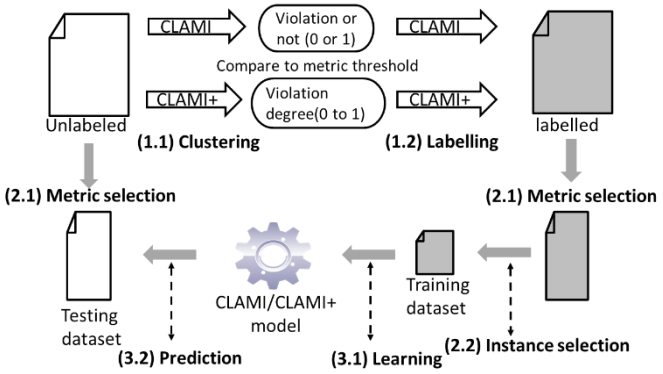

Figure 1. The overview of CLAMI/CLAMI+. It consists of three phases with two steps in each phase. The first phase is clustering and labeling (step 1.1 and 1.2), the CLAMI and CLAMI+ are different in this phase. The second phase is metric selection and instance selection (step 2.1 and 2.2). The third phase is learning and prediction (step 3.1 and 3.2).

The goal of our study is to conduct the change-prone class prediction in an automated way without the need of historical data. In our empirical experiments, we evaluate the self-learning methods on 14 open source projects which come from the Qualitas Corpus [17]. As a result, the self-learning methods yields a reasonable performance which improves the typical inter-version prediction models by 5.2%-19.7% (average CCR) and 13.9%-27.2% (average AUC), respectively. In addition, considering the average performance of all datasets, the proposed CLAMI+ method improves the CLAMI method by 2.9% (average CCR) and 3.2% (average AUC). In summary, the contributions of this study are as follows:

- We apply self-learning approach to tackle change-prone class prediction on new projects or projects with limited historical data. To the best of our knowledge, this is the first study to adopt self-learning approach in change-prone prediction. In addition, we propose a novel self-learning method CLAMI+ by extending the method CLAMI.
- We present an empirical study to evaluate the self-learning methods compared with typical inter-version change-prone class prediction methods on 14 public datasets.

II. Related Work

A variety of approaches have been proposed to predict change-prone classes. For example, Amoui et al [12] 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. Godara et al [18] proposed an ID3 prediction model based on multi-factors. Koru et al. [19] 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 [20] 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 [5] proposed an empirical study which used ensemble methods on change prediction. They found that ensemble methods can achieve a better performance than individual models. However, one issue in the above-mentioned works is that the prediction model relies on learning from the historical data in a supervised way, such as learning from the labeled data from previous project version or learning from labeled data within a project version (i.e., cross validation). It is difficult to apply the technique on new projects or projects with limited historical data.

Cross project prediction is a solution to address the above-mentioned limitation. The cross project concept is introduced by Briand et al [21]. It has been widely used in defect prediction [14, 22-25]. In change-prone class prediction, there are also a few studies which have investigated the cross-project change prediction recently.

Malhotra et al [13] proposed to build the cross project change prediction model by using the logitboost method. In another work, Malhotra et al [26] 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 inter-version prediction within a project (i.e., learning from previous version and testing on the current version) [13]. Besides, one main issue remains in cross-project prediction is that different projects possess different data distributions [16]. How to select an appropriate source as the training data is a difficult task [13].

To address the above-mentioned limitation, the self-learning method can enable the prediction task which does not need a prior labeled source as the training dataset. In other words, the self-learning method leads to building the change prediction model through learning by itself.

III. Approach

This section describes the process of the self-learning approach. It consists of three phases and we describe the three phases in three subsections (subsection A, B and C). 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.

A. *Clustering and Labeling*

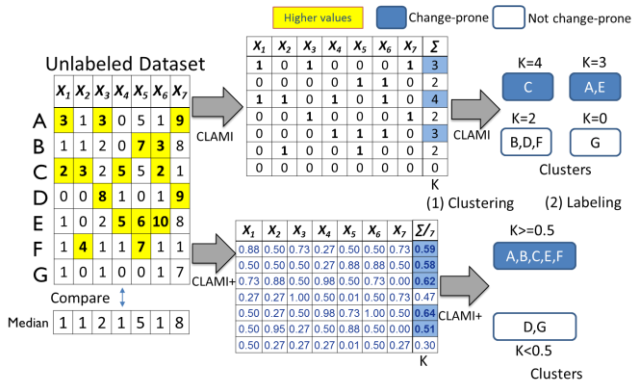

Figure 2. The process of the first phase in the self-learning approach. Higher values mean the metric value is greater than the median.

1) Clustering: The key idea of the clustering process in the self-learning approach is shown in Figure 2. We use A-G to denote the instances of the dataset and x_1-x_7 denote the adopted metrics. A specific cutoff threshold is set as the median value for each metric as Nam et al [16] described. The first step is to compare the metric value to the threshold value for each metric. 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 Figure 2. After that, the clustering process groups the instances by the sum (K value) of the count of the higher values. For example, the instances A and E belong to one cluster (K=3) which means there are three higher values in A and E. However, one issue remains 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 violated 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 Figure 2 shows. In detail, we adopt the sigmoid function which is often used as the activation function in neural networks to conduct the probabilistic transformation. Formally, suppose there are M instances and N metrics, X_{ij} denotes the *j-th* metric value of the *i-th* instance, N_j denotes the threshold value of the *j-th* metric. The violation degree of the *j-th* metric of the *i-th* instance $P(V_{ij})$ is computed as Formula (1). Different from the CLAMI, the K value in CLAMI+ represents the mean violation degree of an instance and we group the instances by $K >= 0.5$ and $K < 0.5$.

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

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 [16]. 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 Figure 2 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 [16, 27]. Since the typical metrics which are adopted in both defect prediction and change prediction (e.g., OO metrics and general size metrics) represent the complexity of the instance, there is also the similar tendency in change-prone prediction [28, 29] (named as change-prone tendency). For example, Koru et al [28] found that that high-change modules had fairly high places in metric rankings, although not the highest places. Malhotra et al. [29] found that the classes whose metric values exceed a threshold value are change prone. Therefore, we label the top half or the first cluster instances as change-prone.

B. *Metric Selection and Instance Selection*

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

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. 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 Figure 2 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 [30, 31]. It is the final step for generating the training dataset in this self-learning 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 Figure 2 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.

C. 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.

IV. Experimental Design

A. Research Questions

We design two research questions to evaluate this study. One is to evaluate the performance of self-learning method. The other is to evaluate the effectiveness of this proposed novel self-learning method CLAMI+.

- **RQ1:** *Is the prediction performance of the self-learning methods comparable to typical prediction methods based on historical data?* The advantage of the self-learning method over typical prediction methods is that it does not need historical labeled dataset. We will answer this question by comparing the self-learning method and the typical prediction methods on the same target dataset. The difference is that the typical prediction methods learn from historical labeled dataset while our self-learning method learn from itself.
- **RQ2:** *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 used in RQ1.

B. Datasets

We evaluate this study on 14 open source projects which come from the public dataset Qualitas Corpus [17] (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 (used in baselines), percentage of changed instances and the total number of instances in the target version are listed in Table I. The percentage and the number of instances possess a substantial range which can validate the model ability among a wide range.

TABLE I: 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

Considering the code metrics, we adopt the typical metrics which are identical with the relevant studies of change prediction [1, 5, 11, 32] as the Table II shows. In detail, five Chidambar and Kemerer metrics [33]: WMC, DIT, NOC, RFC, and LCOM; four Li and Henry metrics [34]: 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.

TABLE II: Summary of The Adopted Metrics in This Study

Metric	Description
WMC	Count of methods implemented within a 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	The average percentage of methods in a class using each data field in the class subtracted from 100 %
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

C. Experimental Baselines

In RQ1, we set the typical inter-version prediction methods [13] within a project as baselines to compare with the self-learning methods. In other words, in order to predict the change-prone classes of the target version, the prediction models of the baselines are built by learning from the labeled dataset of the previous release version. In our experiment, the target version is as Table I shows and we set the previous neighbor version of the target version as the training source in the baselines. The first baseline is the change-prone class prediction based on logitboost (LB) which is proposed by Malhotra et al [13]. In addition, to avoid the bias from only one method, we also adopt three typical machine learners as baselines which are used in all three empirical studies on

TABLE III: PERFORMANCE COMPARISON BETWEEN SELF-LEARNING METHODS AND FOUR INTER-VERSION PREDICTION BASELINES. IF THE PERFORMANCE OF THE SELF-LEARNING METHODS CLAMI/CLAMI+ OUTPERFORMS ALL THE FOUR BASELINES, THE RESULTS ARE IN BOLD. THE BETTER RESULTS BETWEEN CLAMI AND CLAMI+ ARE UNDERLINED.

Project	CCR						AUC					
	LB	MLP	RBF	SVM	CLAMI	CLAMI+	LB	MLP	RBF	SVM	CLAMI	CLAMI+
'ant-1.8.2.0'	88.63	89.22	88.63	87.80	58.29	58.29	0.60	0.60	0.58	0.51	**0.65**	**0.65**
'antlr-3.4.0'	56.85	53.53	46.47	36.51	**65.56**	**65.56**	0.63	0.60	0.55	0.47	**0.65**	**0.65**
'argouml-0.32.2'	60.33	60.33	60.33	60.33	46.05	52.23	0.51	0.51	0.51	0.51	0.49	**0.52**
'azureus-4.1.0.4'	92.32	92.38	92.29	92.29	52.83	52.83	0.47	0.47	0.47	0.47	**0.64**	**0.64**
'freecol-0.10.5'	29.93	30.10	28.26	28.26	**63.88**	**64.05**	0.47	0.47	0.45	0.45	**0.67**	**0.67**
'freemind-0.6.7'	39.19	43.24	32.43	24.32	**52.70**	**60.81**	0.66	0.62	0.56	0.58	0.61	0.63
'hibernate-3.1.2.0'	6.92	8.22	6.38	6.38	**50.70**	**57.62**	0.54	0.55	0.54	0.54	**0.61**	**0.62**
'jgraph-5.12.1.0'	84.91	86.79	88.68	83.02	69.81	69.81	0.65	0.67	0.73	0.49	0.72	**0.77**
'jmeter-2.8.0.0'	52.89	57.59	51.08	50.84	**63.98**	**66.63**	0.56	0.60	0.56	0.58	**0.65**	**0.67**
'jstock-1.0.7.2'	89.49	89.49	89.13	88.41	55.07	55.07	0.61	0.61	0.60	0.57	**0.62**	**0.66**
'jung-1.7.4'	72.65	72.01	71.58	71.15	52.35	51.50	0.49	0.48	0.47	0.47	**0.51**	**0.56**
'junit-4.10.0'	12.88	14.72	9.20	7.98	**52.76**	**52.76**	0.57	0.59	0.55	0.55	**0.76**	**0.76**
'lucene-4.0.0.0'	34.35	34.35	34.35	34.35	**65.97**	**65.97**	0.47	0.47	0.47	0.47	**0.65**	**0.65**
'weka-3.5.8'	36.10	33.87	37.62	21.00	**55.59**	**55.67**	0.52	0.48	0.53	0.45	**0.59**	**0.58**
Average	54.10	54.70	52.60	49.47	**57.54**	**59.20**	0.55	0.55	0.54	0.51	**0.63**	**0.65**

change prediction proposed by Elish et al [5]. Therefore, the second, third and the fourth baseline is Multilayer perceptron (MLP), Radial basis function network (RBF) and Support vector machine (SVM), respectively.

In RQ2, we compare the extended CLAMI+ to the original CLAMI method proposed by Nam et al [16].

D. Performance Measures

Same as in the work of Elish et al [5], 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.

V. RESULTS

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

Overall, in terms of RQ1, the self-learning methods CLAMI and CLAMI+ show comparable performance to the four inter-version prediction methods. In particular, considering the CCR measure, the self-learning methods outperform the four baselines among 8 datasets. In the dataset like ant-1.8.2.0, the self-learning methods perform worse. However, considering the average of all datasets, self-learning methods CLAMI/CLAMI+ improve them by 5.2%-19.7%. Considering the AUC measure, self-learning methods CLAMI/CLAMI+ outperform four baselines among 11 datasets and improve them by 13.9%-27.2% in average of all datasets. Note that the self-learning methods do not need prior labeled data but achieve comparable performance than inter-version prediction methods. In terms of RQ2, 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 or CCR. 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 and 3.2% in AUC.

In addition, we conduct the Friedman test on the performance comparison when comparing multiple methods as suggested by Demša [35]. The Friedman test compares whether the difference of the average ranks of the performance of the methods are statistically significant or not. We translate the question into the null hypothesis H_{null}: There is no significant difference between the average ranks of the performance of all the methods. And the alternative hypothesis H_{alt} is that there is a significant difference between the average ranks of all the methods. Table IV shows the Friedman test results. We provided the average ranks (the approach with the best performance is ranked in "6") and the significant level *p-value*. In terms of CCR, the CLAMI and CLAMI+ show a comparable ranks although not the best. In terms of AUC, the CLAMI and CLAMI+ show the higher ranks than other four methods and the CLAMI+ is the best. Note that the *p-values* in both of the two measures are less than 0.05 which enable us to reject the null hypothesis and accept the alternative hypothesis. This indicates that there is a statistical significant difference between the average ranks of all the methods in the two performance measures.

TABLE IV: FRIEDMAN TEST FOR THE PERFORMANCE COMPARISON

Measure	Average rank						*p-value*
	LB	MLP	RBF	SVM	CLAMI	CLAMI+	
CCR	3.86	4.25	3.14	2.18	3.61	3.96	0.0356
AUC	3.18	3.29	2.32	1.79	4.79	5.64	0.0000

VI. THREATS TO VALIDITY

Impact of the threshold. Threshold decides the results of the clustering and initialized labeling phase. There are various methods to decide a metric threshold. However, we did not provide the analysis on the impact of different thresholds. This might be a threat to our work. In this work, we adopt a typical threshold (i.e., the median) to mitigate this issue. A more

refined work is to take into account the effects of different thresholds.

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. There are several parameters in a sigmoid function, such as dynamic range, and slope. 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. Also, we have a plan to conduct additional experiment on the impact of the parameters.

VII. Conclusion and Future Work

In this paper, we proposed to adopt self-learning approach to tackle change-prone class prediction on new projects or projects with limited historical data. Concretely, we applied a state-of-art self-learning method CLAMI and proposed a novel approach CLAMI+ by extending CLAMI on change-prone class prediction. This enables prediction for new projects or projects with limited historical data. The empirical study among 14 open source projects showed that the self-learning methods yield better or comparable results to four typical inter-version prediction methods in terms of CCR and AUC. In addition, the proposed CLAMI+ method slightly improves the CLAMI method on average.

In the future, we plan to enhance the effectiveness of our approach further. Concretely, we plan to investigate the impact of various thresholds, such as mean, standard deviation and different percentiles. In addition, we plan to improve the performance by proposing an adaptive method to determine the optimized parameters of the sigmoid function for different metrics.

Acknowledgments

The work described in this paper was partially supported by the National Natural Science Foundation of China (Grant no. 91118005, 61173131, 11202249), Chongqing Graduate Student Research Innovation Project (grant no. CYS14008 and CYS15022), Program for Changjiang Scholars and Innovative Research Team in University (Grant No. IRT1196) and the Fundamental Research Funds for the Central Universities of China (Grant No. 106112014CDJZR098801).

References

[1] Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," Journal of Systems and Software, vol. 80, pp. 1349-1361, 2007.

[2] R. Malhotra and 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, and J. D. Kymer, "Automatically classifying software changes via discriminative topic model: Supporting multi-category and cross-project," Journal of Systems and Software, vol. 113, pp. 296-308, 2016.

[4] Y. Fu, M. Yan, X. Zhang, L. Xu, D. Yang, and J. D. Kymer, "Automated classification of software change messages by semi-supervised Latent Dirichlet Allocation," Information and Software Technology, vol. 57, pp. 369-377, 2015.

[5] M. Elish, H. Aljamaan, and I. Ahmad, "Three empirical studies on predicting software maintainability using ensemble methods," Soft Computing, vol. 19, pp. 2511-2524, 2015/09/01 2015.

[6] Z. Yuming, H. Leung, and X. Baowen, "Examining the Potentially Confounding Effect of Class Size on the Associations between Object-Oriented Metrics and Change-Proneness," IEEE Transactions on Software Engineering vol. 35, pp. 607-623, 2009.

[7] F. Khomh, M. Di Penta, Gue, x, he, x, et al., "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.

[8] D. Posnett, C. Bird, and P. Dévanbu, "An empirical study on the influence of pattern roles on change-proneness," Empirical Software Engineering, vol. 16, pp. 396-423, 2011/06/01 2011.

[9] M. O. Elish and M. Al-Rahman Al-Khiaty, "A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software," Journal of Software: Evolution and Process, vol. 25, pp. 407-437, 2013.

[10] S. Eski and 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.

[11] C. van Koten and A. R. Gray, "An application of Bayesian network for predicting object-oriented software maintainability," Information and Software Technology, vol. 48, pp. 59-67, 2006.

[12] M. Amoui, M. Salehie, and L. Tahvildari, "Teporal software change prediction using neural networks," International Journal of Software Engineering and Knowledge Engineering, vol. 19, pp. 995-1014, 2009.

[13] R. Malhotra and 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.

[14] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," presented at the Proceedings of the International Conference on Software Engineering, San Francisco, CA, USA, 2013.

[15] A. Panichella, R. Oliveto, and 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.

[16] J. Nam and 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.

[17] E. Tempero, C. Anslow, J. Dietrich, T. Han, L. Jing, M. Lumpe, et al., "The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies," in Proceedings of the 17th Asia Pacific Software Engineering Conference, 2010, pp. 336-345.

[18] D. Godara and R. Singh, "A New Hybrid Model for Predicting Change Prone Class in Object Oriented Software," International Journal of Computer Science and Telecommunications, vol. 5, pp. 1-6, 2014.

[19] A. Güneş Koru and H. Liu, "Identifying and characterizing change-prone classes in two large-scale open-source products," Journal of Systems and Software, vol. 80, pp. 63-73, 2007.

[20] H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen, "The ability of object-oriented metrics to predict change-proneness: a meta-analysis," Empirical Software Engineering, vol. 17, pp. 200-242, 2012/06/01 2012.

[21] L. C. Briand, W. L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," IEEE Transactions on Software Engineering, vol. 28, pp. 706-720, 2002.

[22] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," Automated Software Engineering, vol. 19, pp. 167-199, 2012/06/01 2012.

[23] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in Proceedings of the 10th IEEE Working Conference on Mining Software Repositories, 2013, pp. 409-418.

[24] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," presented at the Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Amsterdam, The Netherlands, 2009.

[25] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction" Empirical Software Engineering, vol. 14, pp. 540-578, 2009.

[26] R. Malhotra and 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.

[27] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," IEEE Transactions on Software Engineering, vol. 33, pp. 2-13, 2007.

[28] A. G. Koru and J. Tian, "Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products," IEEE Transactions on Software Engineering vol. 31, pp. 625-642, 2005.

[29] R. Malhotra and A. Bansal, "Prediction of Change Prone Classes using Threshold Methodology," Advances in Computer Science and Information Technology, vol. 2, pp. 30-35, 2015.

[30] E. Kocaguneli, T. Menzies, J. Keung, D. Cok, and R. Madachy, "Active learning and effort estimation: Finding the essential content of software effort estimation data," IEEE Transactions on Software Engineering, vol. 39, pp. 1040-1053, 2013.

[31] Y. F. Li, M. Xie, and T. N. Goh, "A study of project selection and feature weighting for analogy based software cost estimation," Journal of Systems and Software, vol. 82, pp. 241-252, 2009.

[32] M. O. Elish and 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.

[33] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on Software Engineering, vol. 20, pp. 476-493, 1994.

[34] W. Li and S. Henry, "Object-Oriented metrics that predict maintainability," Journal of Systems and Software, vol. 23, pp. 111-122, 1993.

[35] J. Demsar, "Statistical Comparisons of Classifiers over Multiple Data Sets," J. Mach. Learn. Res., vol. 7, pp. 1-30, 2006.