CLAMI: Defect Prediction on Unlabeled Datasets

Jaechang Nam and Sunghun Kim
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology, Hong Kong, China
Email: {jcnam,hunkim}@cse.ust.hk

***Abstract*—Defect prediction on new projects or projects with limited historical data is an interesting problem in defect prediction studies. This is largely because it is difficult to collect defect information to label a dataset for training a prediction model. Cross-project defect prediction (CPDP) has tried to solve this problem by reusing prediction models built by other projects that have enough historical data. However, CPDP does not always build a strong prediction model because of the different distributions among datasets. Approaches for defect prediction on unlabeled datasets have also tried to address the problem by adopting unsupervised learning but it has one major limitation, the necessity for manual effort.**

In this study, we propose novel approaches, CLA and CLAMI, that show the potential for defect prediction on unlabeled datasets in an automated manner without need for manual effort. The key idea of the CLA and CLAMI approaches is to label an unlabeled dataset by using the magnitude of metric values. In our empirical study on seven open-source projects, the CLAMI approach led to the promising prediction performances, 0.636 and 0.723 in average f-measure and AUC, that are comparable to those of defect prediction based on supervised learning.

I. Introduction

Defect prediction plays an important role in software quality [1]. Defect prediction techniques provide a list of defect-prone source code so that quality assurance (QA) teams can focus on the most defective parts of their products in advance. In this way, QA teams can effectively allocate limited resources on which to review and test their software products before releasing them. In industry, defect prediction techniques have been actively adopted for software quality assurance [2], [3], [4], [5], [6].

Researchers have proposed and facilitated various defect prediction algorithms and metrics [1], [4], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. Most defect prediction models are based on supervised learning for classification (e.g., predicting defect-proneness of a source code file) or regression (e.g., predicting the number of defects in a source code file) [1], [17], [18], [19]. Rather than using a machine learning technique, Kim et al. proposed *BugCache*, which manages defect-prone entities in source code by adapting the cache concept used in operating system [12]. Metrics for defect prediction can be divided into code and process metrics [20]. Code metrics represent how the source code is complex while process metrics represent how the development process is complex [1], [20].

In particular, studies on defect prediction metrics have been actively conducted as the use of software archives such as version control systems and issue trackers has become popular [20]. Most metrics proposed over the last decade such as change/code metric churn, change/code entropy, popularity, and developer interaction have been collected from various software archives [7], [9], [10], [11], [13], [16], [17].

However, typical defect prediction techniques based on supervised learning are designed for a single software project and are difficult to apply to new projects or projects that have limited historical data in software archives. Defect prediction models based on supervised learning can be constructed by using a dataset with actual defect information, that is, *a labeled dataset*. Bug information usually accumulates in the software archives, thus new projects or projects with a short development history do not have enough defect information. This is a major limitation of the typical defect prediction techniques based on supervised learning.

To address this limitation, researchers have proposed various approaches to enable defect prediction on projects with limited historical data. Cross-project defect prediction that builds a prediction model using data from other projects has been studied by many researchers [19], [21], [22], [23], [24], [25], [26], [27]. Defect prediction techniques on unlabeled datasets were proposed as well [28], [29]. Recently, an approach to build a universal defect prediction model by using multiple project datasets was introduced [30].

However, there is still an issue of different distributions among datasets in existing approaches for cross-project defect prediction (CPDP) and universal defect prediction (UDP). In CPDP and UDP, the major task is to make the different distributions of datasets similar since prediction models work well when the datasets for training and testing a model have the same distributions [31], [32]. However, the approaches based on making the different distributions similar between training and test datasets may not always be effective [22], [23].

Compared to CPDP and UDP, the existing approaches for defect prediction on unlabeled datasets are relatively less affected by the issue of different distributions among datasets but will always require manual effort by human experts [28], [29]. Zhong et al. proposed the expert-based defect prediction on unlabeled datasets where a human expert would label clusters from an unlabeled dataset after clustering [29]. Catal et al. proposed the threshold-based approach where labeling datasets is conducted based on a certain threshold value of a metric [28]. A proper metric threshold for the threshold-based approach is decided by the intervention of human experts [28]. Since these approaches on unlabeled datasets are conducted on the test dataset itself, the issue of the different distributions among datasets is not affected [28], [29]. However, these approaches require manual effort by human experts [28], [29].

The goal of this study is to propose novel approaches, CLA and CLAMI, which can conduct defect prediction on

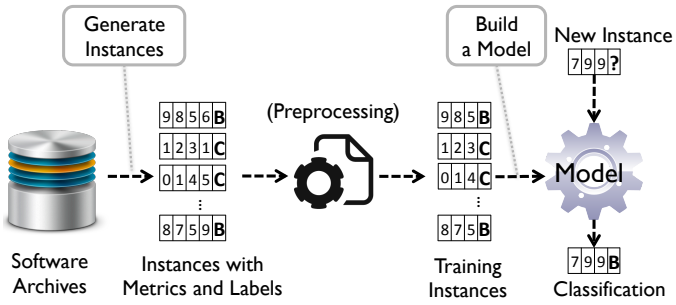

Fig. 1: The typical defect prediction process based on supervised learning.

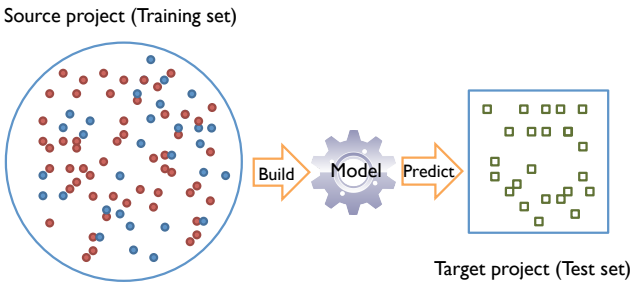

Fig. 2: Cross-Project Defect Prediction

unlabeled datasets in an automated manner. The key idea of the CLA/CLAMI approaches is to label an unlabeled dataset by using the magnitude of metric values.

In our empirical study, the CLA/CLAMI approach led to promising prediction performances of 0.636 (average f-measure) and 0.723 (average AUC), while the typical defect prediction based on supervised learning showed 0.585 and 0.694 in average f-measure and AUC respectively. In addition, the CLA/CLAMI approaches outperformed or were comparable to the existing approaches for defect prediction on unlabeled datasets with statistical significance. These promising results show that our CLA/CLAMI approaches have the potential for defect prediction on new projects that have limited historical data.

The contributions of this study are as follows:

- Proposing novel approaches, CLA and CLAMI, for defect prediction on unlabeled datasets in an automated manner.
- An empirical study to evaluate the CLA/CLAMI approaches against existing defect prediction approaches.

II. Background and Related Work

A. Typical Defect Prediction Process

Fig. 1 shows the typical defect prediction process for a single software project based on supervised machine learning (classification). Since this whole process is conducted 'within' the single software project, it is called within-project defect prediction (WPDP).

In this process, we first collect various data such as development artifacts and historical information from software archives such as the version control system and the issue tracker of the software project. Using the data from software archives, we can measure the complexity of the software project and its development process. The complexity measurement can be conducted at different granularities such as the function (method), file (class), or subsystem (package) levels. In addition, we can collect defect information for the software project from the issue tracker and commit messages in the version control system.

With the collected data, we can generate instances that consist of metrics (features in a machine learning sense) and labels. In Fig. 1, an entity with numeric values and labels, 'B' and 'C', is called an instance. Each instance represents a function, a file, or a subsystem according to their granularity. Metrics measure the complexity of the software or its development process and each metric has a numeric value in the instance. There are various kinds of metrics such as lines of code, the number of functions, and the number of authors touching a source code file [1], [16]. The instance is also labeled as either buggy (B) or clean (C) by using the collected defect information as in Fig. 1.

With the set of labeled instances, we can apply some preprocessing techniques used in machine learning. The representative preprocessing techniques are normalization and feature selection and are widely used in defect prediction studies [1], [22], [23], [33]. However, preprocessing may not be necessary depending on the defect prediction models [10], [13], [22].

We can then generate a dataset of training instances to build a prediction model. Since the prediction model is built by using supervised learning, we can use various machine learners to build a classification model such as Logistic Regression, Decision Tree, Naive Bayes, and Random Forest [10], [13], [22], [34], [35], [36].

Using the prediction model built by the dataset of training instances, we can predict new instances as buggy or clean. For example, as shown in Fig. 1, the new instance marked with '?' is classified as buggy by the prediction model.

This typical defect prediction for single projects, WPDP, has a major limitation. For any new projects or projects with limited historical data, it is difficult to build defect prediction models since we cannot generate the labeled dataset for training a model without defect information [18], [22]. In Section II-B, we list related studies addressing this limitation.

B. Defect Prediction on Projects with Limited Historical Data

Researchers have tried to address the issue on new projects or projects with limited historical data to build defect prediction models by proposing techniques for cross-project defect prediction [21], [22], [23], [24], [26], [30], [37] and defect prediction on unlabeled datasets [28], [29].

1) Cross-Project Defect Prediction (CPDP): Fig. 2 shows the typical CPDP process. In Fig. 2, the small shaded circles in *Source* project represent labeled instances and the small rectangles in *Target* project represent unlabeled instances. We first build a prediction model using the source project with labeled instances. Then, using the model, we can predict

whether an instance in the target project is defect-prone or not.

However, CPDP has a challenging issue that prediction performance was not practical [18]. Zimmermann et al. conducted 622 cross predictions but only 21 predictions were successful in their experimental setting [18]. Watanabe et al. proposed metric compensation approach to improve a CPDP model. The approach by Watanabe et al. makes a target project similar to a source project by normalizing metric values using the average metric values [24]. The CPDP performance with metric compensation improved that without metric compensation but was worse than WPDP [24]. Turhan et al. proposed the nearest neighbour (NN) filter for CPDP [23]. The NN filter selects the 10 nearest source instances for each target instance [23]. In other words, when building a prediction model, the NN filter approach uses the most similar source instances to the target instances [23]. However, its prediction performance is still worse than that of WPDP [23].

To resolve the CPDP issue, Ma et al. and Nam et al. facilitated transfer learning techniques from the machine learning community [21], [22]. Ma et al. proposed Transfer Naive Bayes (TNB) which provides more weight to the source instances that are similar to target instances when building a Naive Bayes model [21]. The TNB led to better prediction performance than the approach based on the NN filter [21]. Nam et al. adopted transfer component analysis (TCA) which is a state-of-the-art transfer learning technique and proposed TCA+ for CPDP [22]. In the empirical study of Nam et al., the performance of CPDP was comparable to that of WPDP [22].

Recently, CPDP models have been evaluated with a view to its cost-effectiveness [25], [26], [27]. Rahman et al. confirmed that CPDP models can outperform WPDP models in terms of cost-effectiveness. Canfora et al. proposed multi-objective approach for CPDP [26]. Multi-objective models built using a genetic algorithm help software engineers choose prediction models having different objectives such as high recall or low cost [26]. In their empirical study, multi-objective models achieved better prediction results than a WPDP model in terms of cost-effectiveness [26]. Panichella et al. proposed a combined defect predictor (CODEP) for CPDP [27]. CODEP considers predicted defects by combining prediction results of different machine learning algorithms and led to better prediction performance than a single prediction model in AUC and cost-effectiveness [27].

Zhang et al. addressed the CPDP issue by proposing the universal defect prediction model [30]. Since the individual project may have its specific defect characteristic, the universal model may not work for all projects [30]. To resolve this limitation, Zhang et al. proposed context-aware rank transformations that change metric values ranged from 1 to 10 across all projects [30]. In this way, the universal model could be built using 1,398 projects from SourceForge and Google code. In their experimental setting it showed a comparable prediction performance to WPDP [30].

The related studies about CPDP actually addressed the same issue we resolve in this study, that is, defect prediction on unlabeled dataset. However, different from our CLA/CLAMI approaches, the CPDP approaches always require abundant source project datasets and the prediction model can be constructed when both source and target projects have the same metric set.

Most studies for CPDP try to make the different distributions of source and target datasets similar by using techniques such as transforming metric values [24], [30], selecting similar instances [23], and using transfer learning [21], [22]. However, those techniques cannot always effectively make different distributions similar when compared to the case where the same project dataset is used. Therefore, they can still suffer from the dataset shift problem [31]. Recently, this problem has also been observed even in different releases of the same project [38].

CLA and CLAMI do not need any techniques to make different distributions of datasets similar since we just use the same project dataset where we want to predict defects. In other words, the CLA/CLAMI approaches do not need to consider any source projects and the limitation of different metric sets in project datasets to build a prediction model.

2) Defect Prediction on Unlabeled Datasets: There are a couple of studies for defect prediction on unlabeled datasets [28], [29].

Zhong et al. proposed the expert-based approach [29]. The expert-based approach first clusters unlabeled instances using a clustering algorithm such as K-means, then asks a human-expert whether a cluster is defect-prone or not, after providing average metric values of the cluster, that is, centroid [29]. Using the expert-based approach, Zhong et al. achieved a 12.08% false positive rate and a 31.13% false negative rate in the best cases.

The expert-based approach always requires human experts to decide whether a cluster is defect-prone or not. Thus, this approach cannot fully automate defect prediction on unlabeled datasets. However, our approaches, CLA and CLAMI, need no human experts and can automate the defect prediction process on unlabeled datasets.

Catal et al. proposed a one-stage threshold based approach [28]. After initially proposing a two-stage approach based on both clustering and threshold [28], they concluded the one-stage threshold based approach is easier than the two-stage approach and still effective enough [28]. The threshold-based approach predicts an instance as buggy when any metric value is greater than the given metric threshold values [28]. The threshold values were decided based on 'experience and hints from the literature', past defect-prone modules, and analysis of past versions of a project [28]. The one-stage threshold-based approach achieved a 32.14% false postive rate and a 20% false negative rate [28].

The threshold-based approach needs to decide metric threshold values in advance. In other words, additional effort is required for metric threshold values. However, CLA and CLAMI do not need additional effort and can build a prediction model using only unlabeled datasets.

III. Approach

Fig. 3 shows the overall process of our CLA and CLAMI approaches for defect prediction on an unlabeled dataset. The key idea of our approaches is to label unlabeled instances by using the magnitude of metric values.

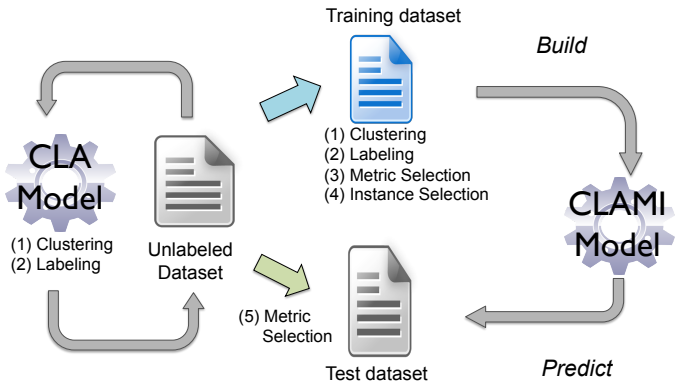

Fig. 3: The overview of CLA and CLAMI for defect prediction on an unlabeled dataset. We name our approaches CLA or CLAMI based on the two or four steps respectively.

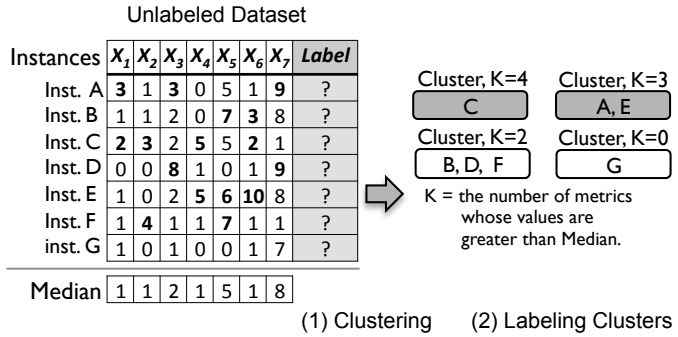

Fig. 4: Clustering and Labeling. Any metric values greater than the median values are in bold font.

The first two steps for CLA and CLAMI are (1) **C**lustering instances and (2) **La**beling instances in clusters. With these two steps, we can label/predict all instances in the unlabeled dataset. CLA consists of only these two steps. Based on the two steps, we name our first approach **CLA**.

CLAMI has two additional steps to generate a training dataset, (3) **M**etric selection, and (4) **I**nstance selection. Based on all four steps, we name our second approach **CLAMI**. In the machine learning community, metric (feature) selection and instance selection have been widely used to improve prediction models by removing noisy metrics and instances [39]. In addition, metric selection and instance selection has also been applied to improving various prediction models in the software engineering community [33], [40], [41], [42], [43].

For CLAMI, (5) we also select metrics from the unlabeled dataset to generate a test dataset with the same set of metrics as in the training dataset generated by CLAMI since both training and test datasets should have the same metric set. The test dataset includes all the instances from the original unlabeled dataset but with only selected metrics. When the training and test datasets are ready for CLAMI, we can build a prediction model using various machine learning classifiers and conduct defect prediction for the unlabeled test dataset.

The following subsection describes the four steps in detail.

A. Steps for CLA and/or CLAMI

1) Clustering Instances: Fig. 4 shows how to cluster instances in a dataset. In Fig. 4, X_1–X_7 represent metrics of the unlabeled dataset, while Inst. A–G represent instances of the dataset. We first identify higher metric values by using a specific cutoff threshold. In the example of Fig. 4, we use a median value for each metric as the threshold to decide higher metric values.[1] For example, the median of the metric, X_1, is 1 from {0,1,1,**1**,1,2,3}. Using the median of the metric values, we can count the number of metrics whose values are greater than the median values in each instance. For example, in the instance A, the number (K) of metrics whose values are greater than the median values is 3 (X_1,X_3, and X_7). In Fig. 4, all higher metric values that are greater than a corresponding median value are in bold font. After computing K values of all instances, we can group instances that have the same K value. As shown in Fig. 4, we can form four clusters with K=4,3,2, and 0 respectively.

2) Labeling: By considering K values, we divide clusters into two groups, a top half and a bottom half, and label the instances in the top half of the clusters as buggy and the others as clean. In Fig. 4, the instances A, C, and E in the clusters, K=4 and K=3, are labeled as buggy and other instances are labeled as clean.

The intuition of this labeling process is based on the defect-proneness tendency of typical defect prediction datasets, that is, *higher complexity causes more defect-proneness* [1], [10], [20]. In other words, since typical defect prediction metrics measure the complexity of the source code and development process, there is a tendency that buggy instances have higher metric values than clean instances [1], [7], [9], [10], [11], [16], [17], [20], [44].

In this sense, we label instances in the top half of the clusters as buggy since the number of the higher metric values in the instances of the top half of the clusters is more than that of the bottom half of the clusters.

3) Metric Selection: After labeling the instances in the top and bottom clusters, we conduct metric selection based on the metric violation scores (MVS) for CLAMI. Since the quality of defect prediction models is highly dependent on the quality of the metrics, metric selection to choose the most informative metrics for prediction models has been widely adopted in defect prediction [33], [40], [41]. Metric selection in CLAMI is based on removing metrics that can minimize violations in the defect-proneness tendency of defect datasets [1], [10], [20]. Not all metrics follow the defect-proneness tendency so that metric selection of CLAMI can be helpful to build a better prediction model.

A violation is a metric value that does not follow the defect-proneness tendency. The metric value (1) in X_1 of the instance E labeled as buggy is not greater than the median (1) so this case is counted as a violation. The metric value (**4**) of X_2 in the instance F labeled as clean is greater than its median value (1) so this case is also considered to be a violation. By counting all these violations, we can define the metric violation score

[1]In Section VI, we apply various cutoff thresholds to decide higher metric values and compare prediction performances on the various thresholds.

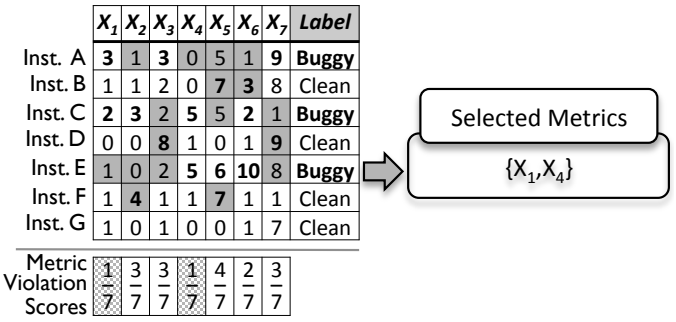

Fig. 5: Computing metric violation scores (MVS) and metric selection. Violated metric values are shaded in dark gray. The metrics with the minimum MVS are selected.

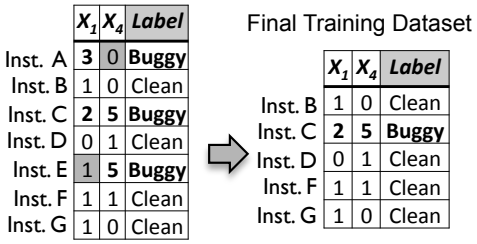

Fig. 6: Instance selection and the final training dataset. Violated metric values are shaded in dark gray. Instances without violations are selected.

of an i-th metric (MVS_i) as follows:

$$MVS_i = \frac{C_i}{F_i} \quad (1)$$

, where C_i is the number of violations in the i-th metric and F_i is the number of metric values in the i-th metric.

Fig. 5 shows how to compute MVS and how to select metrics by MVS. In Fig. 5, shaded metric values represent violations. For example, in X_3, the number of violations is '3'. By equation (1), the violation score of X_3, MVS_3, is $\frac{3}{7}$(≈0.429). In this way, we can compute the violation scores of all the metrics. Then, we can select metrics whose violation scores are minimum. In Fig. 5, metrics that have the minimum MVS, that is, $\frac{1}{7}$, are X_1 and X_4. Thus, we finally select X_1 and X_4 as metrics for the training dataset.

4) Instance Selection: Fig. 6 shows the final step to generating a training dataset for CLAMI, that is, instance selection. After the metric selection, there may still be violations in the metric values. We remove instances that have any violated metric value. As shown in Fig. 6, the instances A and E have violations in their values, thus we remove instances A and E to generate the final training dataset. Instance selection is widely used in various prediction models as well [45], [46], [42], [47].

After instance selection, it might be that there are no buggy and/or clean instances because of many violations. It is not possible to build defect prediction models based on supervised learning when both buggy and clean instances do not exist together. In this case, we can use the next minimum MVS to select metrics until we can generate a training dataset with both buggy and clean instances together.

TABLE I: The seven defect datasets from two groups.

Group	Dataset	# of instances		# of metrics	Prediction Granularity
		All	Buggy(%)		
NetGene [44]	Httpclient	361	205(56.79%)	465	File
	Jackrabbit	542	225(41.51%)		
	Lucene	1671	346(10.71%)		
	Rhino	253	109(43.08%)		
ReLink [48]	Apache	194	98(50.52%)	26	File
	Safe	56	22(39.29%)		
	ZXing	399	118(29.57%)		

The reason we propose CLAMI models that are built by a machine learner is to get advantage from metric and instance selection. CLA can label all instances. However, CLA may have violated metrics and instances that do not follow the defect-proneness tendency of typical defect prediction metrics. In this reason, CLA may label instances incorrectly. To minimize the instances that might be incorrectly labeled, we apply metric and instance selection by removing the violations. Then, we predict defects of the unlabeled instances by using the machine learner built using a training set that consists of most representative metrics and instances selected by CLAMI.

IV. Experimental Setup

A. Research Questions

To evaluate the CLA/CLAMI approaches, we set the following research questions.

- RQ1: Are the prediction performances of CLA/CLAMI comparable to those of typical defect prediction based on supervised learning for a single software project?
- RQ2: Are the prediction performances of CLA/CLAMI comparable to those of existing approaches for defect prediction on unlabeled datasets?

In RQ1, we first compare CLA/CLAMI to the typical defect prediction using labeled data, that is, supervised learning. If CLA/CLAMI shows comparable prediction results to the typical defect prediction, then in practice it can be used for projects without labeled data.

We also compare CLA/CLAMI to the existing defect prediction approaches on unlabeled datasets. In RQ2, CLA/CLAMI is compared with two baselines, that is, threshold-based and expert-based approaches [28], [29].

B. Benchmark Datasets

Table I lists seven datasets from two groups, NetGene [44] and ReLink [48], used in our empirical study.[2] The prediction performance of defect prediction models are significantly affected by noisy defect data [43]. The noisy defect data are usually caused when the defects are collected by automatic tools for mining software archives [48], [43]. For this reason, we choose these seven datasets as our experimental subjects since the defect data in the datasets are manually verified or linked to code changes [44], [48].

[2]Herzig et al. [44] used the combined metrics of Network [19] and Genealogy in their empirical study. We call this dataset group NetGene, after several letters of two names.

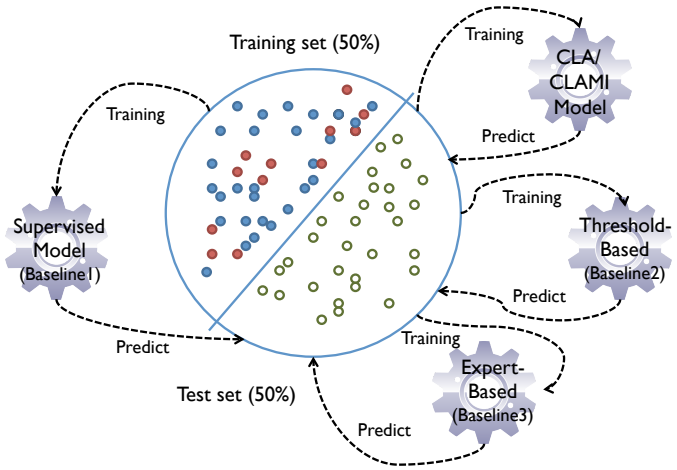

Fig. 7: Experimental setup for baselines and CLA/CLAMI.

The NetGene datasets from the study by Herzig et al. [44] were generated by using more than 7,400 issue reports manually classified. Herzig et al. discussed the issue about misclassified issue reports [49]. Since the misclassified issue reports provide wrong labels to defect datasets, it is critical for defect prediction models [49]. Thus, for our empirical study, we used all the NetGene datasets used in the study by Herzig et al. [44].

The NetGene datasets include network and change genealogy metrics [44]. Herzig et al. proposed metrics based on change genealogies and validated the metrics with existing metrics such as complexity metrics and network metrics [44]. In their empirical study, the combined metrics using network and change genealogy metrics led to the best prediction performance in their empirical study [44].

We also used the golden datasets of ReLink as experimental subjects [48]. Bird et al. discussed the problem about missing links between issue reports and change logs that impact on defect prediction performance [50]. Wu et al. addressed the missing link issue by ReLink and used golden datasets based on manually inspecting links in their empirical study [48]. Each ReLink dataset consists of 26 code complexity metrics extracted by the Understand tool [48], [51].

C. Experimental Design with Baselines

To evaluate CLA/CLAMI, we set three baselines: supervised learning, threshold-based, and expert-based approach.

1) Baseline1: We used supervised learning models trained by labeled data as a baseline to validate RQ1 since the typical defect prediction model for a single software project is based on supervised learning [1], [10], [13], [19], [20], [34], [35], [37]. Fig. 7 shows how to build a supervised learning model as well as a CLA/CLAMI model for a dataset. We compare the prediction performance of a typical defect prediction model based on the supervised learning model (Baseline1) and our CLA/CLAMI model. In supervised learning, we need labeled data, thus to build a supervised learning model, we divided the datasets into two splits as in Fig. 7. In the first prediction round, we build a prediction model using the first split as a training set and test the model on the second split as a test set. Then, for the second round, the first split is used as a test set and the second split is used as a training set. As shown in Fig. 7, the supervised learning model is built using the training set (50%) with labeled instances (small shaded circles) and predicts defects on the test set (50%) with unlabeled instances (small unshaded circles). Since our approaches, CLA and CLAMI, are for an unlabeled dataset, we build CLA/CLAMI models using only the test set with unlabeled instances as in Fig. 7. From both the supervised learning model and the CLA/CLAMI model, we can measure the prediction performance and evaluate whether the prediction performance of our CLA/CLAMI model is comparable to that of the supervised learning model or not (RQ1). Since there is randomness to split a dataset into training and test sets, we repeat the first and second rounds (i.e. two-fold cross validation) 500 times to conduct 1000 predictions and report the averaged results [22], [52].

As a machine learning classifier for both the supervised learning model and CLAMI, we use Logistic regression that has been widely used and shown good prediction performance in defect prediction studies [18], [22], [36], [53], [54], [55]. In addition, we used other classifiers such as Bayesian network, J48 decision tree, Logistic model tree, Naive Bayesian, Random forest, and Support vector machine [10], [13], [22], [34], [35], [56]. We discuss prediction performances on various classifiers in Section VI.

2) Baseline2 and Baseline3: Fig. 7 also shows how to predict defects in an unlabeled dataset using existing approaches based on threshold (Baseline2) and expert (Baseline3) and the CLA/CLAMI models. Since the two baseline approaches and the CLA/CLAMI models are for the unlabeled dataset, each prediction is conducted on the same test set with unlabeled instances.

The threshold-based approach predicts a source code file as buggy when any metric value of the file is greater than the designated values of a threshold vector [28]. Catal et al. set the threshold vector for metrics including lines of code, cyclomatic complexity, unique operator, unique operand, total operator, and total operand as $(65, 10, 25, 40, 125, 70)$ in their study. With this threshold vector, if the cyclomatic complexity of a source code file is greater than the threshold value of 10, the file is predicted to be buggy. Catal et al. actually proposed threshold-based and clustering-based approaches together but opted to use the threshold-based approach since it is easier and more effective than the clustering-based approach [28]. Thus, we use the threshold-based approach as one of baselines from existing approaches for defect prediction on unlabeled datasets.

We generate threshold vectors for each dataset in Table I by using the tuning machine technique as did Catal et al. [28], [57]. Catal et al. used three techniques to determine a threshold vector such as 'experience and hints from the literature', 'tuning machine', and 'analysis of multiple versions' as proposed by Marinescu [28], [57]. However, we only apply the tuning machine technique that decides a threshold value maximizing the number of correctly predicted instances by considering past defects [28], [57] since 'experience and hints from the literature' and 'analysis of multiple versions' are not available for our subject datasets. Past defects on each subject dataset are not available either, therefore we compute threshold values maximizing the number of correctly predicted instances in the

TABLE II: Comparison of results between baselines and CLA/CLAMI in precision, recall, f-measure, and AUC. (SL: Supervised Learning, THD: Threshold-based, EXP: Expert-based) The better CLA/CLAMI results that SL, THD, and EXP with statistical significance (Wilcoxon signed-rank test, $p<0.05$) are in bold font, in gray, and with an asterisk (*) respectively. The better results between CLA and CLAMI with statistical significance are underlined.

Project	Precision					Recall					F-measure					AUC	
	SL	THD	EXP	CLA	CLAMI	SL	THD	EXP	CLA	CLAMI	SL	THD	EXP	CLA	CLAMI	SL	CLAMI
Httpclient	0.728	0.990	0.814	**0.759**	**0.774**	0.726	0.037	0.825	0.711	0.677	0.725	0.019	0.818	**0.734**	0.722	0.722	**0.772**
Jackrabbit	0.653	0.649	0.765	0.634	0.631	0.645	0.117	0.634	**0.754***	**0.752***	0.648	0.128	0.689	**0.689**	**0.686**	0.727	**0.751**
Lucene	0.538	0.340	0.619	0.301	0.287	0.483	0.207	0.225	**0.639***	**0.632***	0.508	0.256	0.243	0.409*	0.395*	0.706	0.596
Rhino	0.656	0.831	0.818	**0.724**	**0.727**	0.599	0.349	0.746	**0.764***	**0.777***	0.623	0.069	0.775	**0.743**	**0.750**	0.686	**0.777**
Apache	0.671	0.663	0.794	**0.727**	**0.722**	0.641	0.553	0.717	**0.684**	**0.714**	0.653	0.634	0.750	**0.705**	**0.718**	0.712	**0.753**
Safe	0.611	0.708	0.964	**0.636**	**0.627**	0.621	0.471	0.815	**0.725**	**0.757**	0.603	0.533	0.878	**0.677**	**0.685**	0.699	**0.770**
ZXing	0.455	0.759	0.626	0.378	0.399	0.265	0.085	0.283	**0.570***	**0.651***	0.331	0.033	0.365	**0.454***	**0.494***	0.603	**0.643**
Average Rank	3.714	2.286	1.429	3.714	3.857	3.429	5.000	2.429	2.143	2.000	3.429	4.857	1.929	2.357	2.429	-Rank Sum: 7 +Rank Sum: 21	

same test dataset for this experiment. This means the threshold values computed by the tuning machine should work better on our subject datasets than those computed with the past defect although we only use the tuning machine.

As the second baseline, we use the expert-based approach proposed by Zhong et al. [29]. The expert-based approach is based on clustering and human experts. In other words, this approach first clusters instances in a dataset by using clustering algorithms such as K-means and then human experts decide whether a cluster is buggy or clean [29]. This approach requires human experts therefore it is impossible to automate the whole process. For our empirical study, we cluster instances using the K-means clustering algorithm and then decide upon the label of each cluster as human experts know the actual label of the cluster [29]; if the number of buggy instances in the cluster is greater than that of clean instances, we label the cluster as buggy, otherwise as clean. For the number of clusters, we used 20 as Zhong et al. suggest [29].

D. Measures

To measure the prediction performance of baselines and CLA/CLAMI, we use precision, recall, f-measure and/or the area under the receiver operating characteristic curve (AUC).

F-measure is a widely used prediction measure in defect prediction studies since it represents the harmonic mean of precision and recall [13], [20], [37], [44], [58]. Precision represents the rate of correctly predicted buggy instances among all instances predicted as buggy. Recall measures the rate of correctly predicted buggy instances among all actual buggy instances.

AUC is known to be useful for comparing different prediction models [25], [34], [35], [59]. AUC is plotted using true positive rate (recall) and false positive rate by changing different prediction thresholds [25]. Thus, AUC is a proper measure for comparing the overall prediction performances of different models. Since f-measure is highly affected by class imbalance and prediction threshold, it can make it difficult to fairly compare prediction models [25]. For this reason, AUC is also widely used in the defect prediction literature [25], [34], [35], [59]. The AUC of 0.7 is considered as promising prediction performance [34], [59].

When comparing baselines and CLA/CLAMI models, we report precision, recall, and f-measure. When comparing a supervised learning model and a CLAMI model, we additionally report AUC. Both the supervised learning model and the CLAMI model are based on statistical models so that AUC can be computed by prediction threshold values. For f-measure, we use 0.5 as a prediction threshold as have most defect prediction studies reporting f-measure [13], [20], [44].

For statistical tests, we conduct the Friedman test ($p<0.05$) with the Nemenyi test as a post-hoc test when comparing multiple models [60] over the seven benchmark datasets. After the Friedman test with the Nemenyi test, we report the visual representation of our results following Demšar guidelines [60]. Demšar also suggested using the Wilcoxon signed-rank test when comparing two models [60]. Thus, to compare the supervised learning model and the CLAMI model in AUC, we conduct the Wilcoxon signed-rank test ($p<0.05$). In addition, when comparing two models (a baseline vs CLA/CLAMI) for each dataset, we conduct the Wilcoxon signed-rank test ($p<0.05$) to test if the performances from 1000 predictions between the baseline and the CLA/CLAMI models are different with statistical significance.

V. Results

Table II shows the prediction performances between baselines and the CLA/CLAMI in various measures such as precision, recall, f-measure, and AUC. For SL and CLAMI, we use Logistic regression as a classifier. As a threshold to decide higher metric values in CLA/CLAMI, we use a median value of each metric. For precision, recall, or f-measure, we conduct the Friedman test and report the average ranks for the five approaches such as supervised learning (SL), the threshold-based approach (THD), the expert-based approach (EXP), CLA, and CLAMI as in the last row of Table II [60]. The Friedman test ranks the five approaches in each dataset [60]. For example, in terms of precision, the ranks of the five approaches in Apache are 4 (SL), 5 (THD), 1 (EXP), 2 (CLA), and 3 (CLAMI); the approach with the best precision is ranked in '1'. After computing the ranks for the seven datasets, we can compute the average rank for each approach. The Friedman test compares whether the average ranks are statistically significant [60].

Table II, for each dataset, if the results of CLA/CLAMI are better than those of SL, THD, and EXP with statistical significance (the Wilcoxon signed-rank test, $p<0.05$), the results of CLA/CLAMI are in bold font, in gray, and asterisked

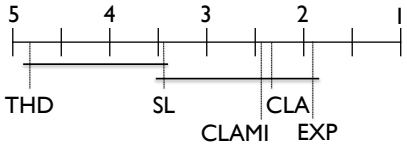

Fig. 8: Comparison of average ranks of baselines and CLA/CLAMI in f-measure. Approaches that are not significantly different (the Nemenyi test, at p = 0.05) are connected.

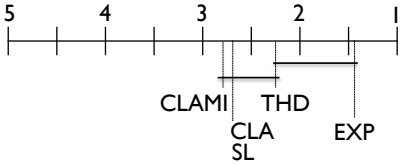

Fig. 9: Comparison of average ranks of baselines and CLA/CLAMI in precision. Approaches that are not significantly different (the Nemenyi test, at p = 0.05) are connected.

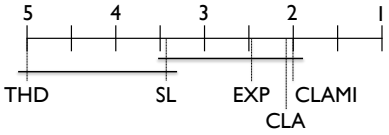

Fig. 10: Comparison of average ranks of baselines and CLA/CLAMI in recall. Approaches that are not significantly different (the Nemenyi test, at p = 0.05) are connected.

(*) respectively. The better results between CLA and CLAMI with statistical significance are underlined. In terms of AUC, the CLAMI results that better SL with statistical significance (the Wilcoxon signed-rank test) are in bold font.

CLA and CLAMI outperform SL and THD in most datasets in terms of f-measure with statistical significance. CLA and CLAMI outperform SL in 6 and 5 datasets respectively and outperform THD in all datasets. In AUC, CLAMI outperforms SL in most datasets except Lucene with statistical significance.

Fig. 8 visualises the results of post-hoc tests by the Nemenyi test after the Friedman test [60]. The Friedman test ($p<0.05$) computes the p-value as 0.004 for f-measure results in Table II. This represents that there are statistical difference between the average ranks of five approaches in terms of f-measure. Then, we conduct the Nemenyi test as a post-hoc test for each pair of the approaches. The top line in Fig. 8 represents the axis where average ranks of five approaches are plotted. Approaches that are not statistically significant are connected. In Fig. 8, there are two groups, (THD, SL) and (SL, CLAMI, CLA, EXP) based on the connected approaches. The lower average rank (the right side in the axis) represents the better prediction performance.

From Fig. 8, we could observe CLA, CLAMI, and EXP outperform THD. CLA, CLAMI, and EXP seem to have an equivalent performance as they are in the same group. SL is on the border between two groups so that it is difficult to conclude whether SL performs the same as other approaches because of insufficient resulting data.

In terms of AUC, CLAMI shows comparable results to SL after conducting the Wilcoxon signed-rank test (p=0.05) between SL and CLAMI for all seven datasets [60]. The Wilcoxon signed-rank test compares the sums of the positive and negative ranks between SL and CLAMI in Table II and the computed p-value is 0.297. Thus, the difference of the rank sums between SL and CLAMI is not statistically significant.

Table II also shows the comparison results between baselines and CLA/CLAMI in precision. For each dataset, CLA and CLAMI outperform SL in four datasets. However, CLA and CLAMI do not outperform THD and EXP in most datasets with statistical significance (only one gray cell against THD and no asterisk against EXP).

Fig. 9 shows the results of the post-hoc tests after the Friedman test (the computed p-value is 0.010) in terms of precision. The average ranks of CLA and CLAMI are worse than EXP with statistical significance. Compared to SL and THD, CLA and CLAMI do not show critical difference as they are grouped together.

Table II shows the comparison results between baselines and CLA/CLAMI in recall. For each dataset, CLA and CLAMI outperform SL and THD in most datasets; results in six datasets are in bold font and all results in CLA and CLAMI are shaded in gray. CLA and CLAMI also outperform EXP in four datasets.

Fig. 10 shows results of the post-hoc tests after the Friedman test (the computed p-value is 0.002) in terms of recall. The average ranks of CLA and CLAMI are better than those of THD with statistical significance. Compared to SL and EXP, CLA and CLAMI do not show critical difference as they are grouped together.

The prediction performances between CLA and CLAMI do not show a significant difference as shown in Fig. 8, 9, and 10. The difference in average ranks between CLA and CLAMI is marginal, e.g., 2.143 vs 2.000 in recall. The average rank (3.714) of CLA in precision is slightly better than that (3.857) of CLAMI but its difference is marginal as well. Since CLA does not require any machine learning classifier, CLA is a simpler approach compared to CLAMI. Thus, we suggest to use CLA. However, in some datasets such as Rhino and Zxing, CLAMI outperforms CLA in precision, recall, and f-measure as in Table II. In this sense, it would be interesting to investigate when CLAMI works better than CLA. We remains this as future work.

Overall, CLA and CLAMI show comparable results to SL (RQ1) and EXP (RQ2) and outperform THD (RQ2) in recall, f-measure, and/or AUC. However, in terms of precision, CLA and CLAMI show the worst ranks although they are not statistically significant against SL and THD. In terms of recall, CLA and CLAMI show the best ranks compared to other approaches although there are no statistical significances against SL and EXP. Menzies et al. already discussed that prediction models with low precision and high recall are useful in many industrial situations [61]. In this sense, CLA and CLAMI that only use a little knowledge about the defect-proneness tendency of metric values show the potential for defect prediction on unlabeled datasets. Note that CLA and CLAMI do not need initially labeled instances and manual

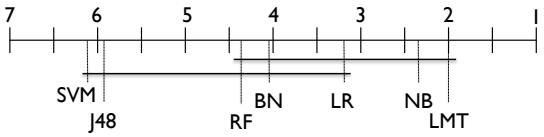

Fig. 11: Comparison of all classifiers against each other in AUC. Approaches that are not significantly different (the Nemenyi test, at p = 0.05) are connected.

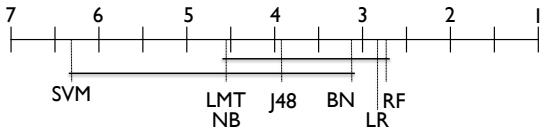

Fig. 12: Comparison of all classifiers against each other in f-measure. Approaches that are not significantly different (the Nemenyi test, at p = 0.05) are connected.

effort but achieve comparable prediction performance to most baselines in terms of recall, f-measure, and AUC.

VI. Discussion

A. Performance on Various Classifiers

We evaluate whether CLAMI models work with other machine learning classifiers. To build CLAMI models, we used Bayesian Network (BN), J48 decision tree (J48), Logistic model tree (LMT), Logistic regression (LR), Naive Bayesian (NB), Random forest (RF), and Support vector machine (SVM) which are widely used in defect prediction [10], [13], [22], [34], [35], [56]. Since we compare multiple classifiers, we conduct the Friedman test with the Nemenyi test.

Fig. 11 visualises the results of post-hoc tests by the Nemenyi test after the Friedman test (the p-value was 0.0005) in AUC [60]. NB and LMT show better average ranks than SVM and J48 in terms of AUC. However, for RF, BN, and LR, it is difficult to conclude that their average ranks are different from other classifiers with statistical significance. The average AUCs are 0.702 (BN), 0.697 (J48), 0.730 (LMT), 0.723 (LR), 0.726 (NB), 0.704 (RF), and 0.656 (SVM). Most AUCs are around 0.700 except SVM.

Fig. 12 shows the results of post-hoc tests by the Nemenyi test after the Friedman test (the p-value was 0.018) in f-measure [60]. The average f-measures are 0.636 (BN), 0.635 (J48), 0.634 (LMT), 0.636 (LR), 0.635 (NB), 0.636 (RF), and 0.534 (SVM). Most f-measures are around 0.635 except SVM.

Ghotra et al. compared various classifiers for defect prediction [56]. SVM was one of the lowest ranked classifiers in their empirical study. In this sense, the low ranks of CLAMI models built by SVM confirm their study [56].

B. Performance on Various Cutoffs

To decide the higher metric values, we apply various cutoff values: n-th percentiles where n is 10, 20,..., 80, and 90 as well as the first and third quartiles (25th and 75th percentile). In total, we used 11 percentiles, P10 (for the 10th percentile), P20, P25 (the first quartile), P30, P40, P50 (median), P60, P70, P75 (the third quartile), P80, and P90.

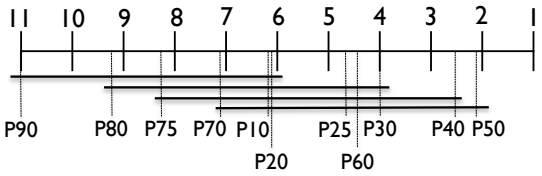

Fig. 13: Comparison of CLAMI models using various cutoffs in f-measure. Approaches that are not significantly different (the Nemenyi test, at p = 0.05) are connected.

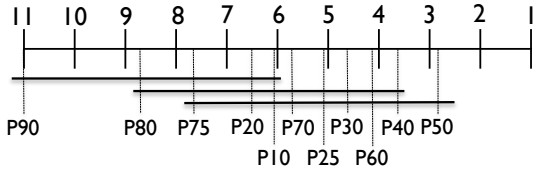

Fig. 14: Comparison of CLAMI models using various cutoffs in AUC. Approaches that are not significantly different (the Nemenyi test, at p = 0.05) are connected.

As shown in Fig. 13 and 14, the median cutoff threshold (P50) shows the best ranks in f-measure and AUC although CLAMI with P50 does not outperform that with most other cutoff thresholds (no statistical significance). CLA shows the similar results on various cutoffs; the P40 (3.000) and P50 (3.571) show the best ranks in f-measure. In this sense, we suggest using a median metric value as the threshold for CLA/CLAMI in the very early stage of the software development phases when there is no information about the best threshold for CLA/CLAMI. However, as EXP results show, human effort is helpful to achieve better prediction performance. In this sense, CLA/CLAMI with additional human effort to decide a proper threshold might lead to better prediction performance as well. Then, the cutoff thresholds for CLA/CLAMI can be properly set by software engineers by using related projects that have similar distributions. Thus, we have a plan to extend CLA/CLAMI models with human effort as future work.

C. Metric Distribution Analysis of Datasets

We investigated whether each metric are correlated with defect-proneness by observing the distributions of metric values of buggy and clean instances. In Fig. 15, the box plots compare the distributions of the metric values of the Safe dataset. Since the Safe dataset has 26 metrics, there are 26 pairs of plots in Fig. 15. A pair of plots shows two distributions of buggy or clean instances for one metric respectively. The distributions of metric values of buggy instances are plotted in gray while those of clean instances are plotted in white. The solid horizontal line in a box represents the median value in each distribution. The top and bottom of boxes represent the third and first quartiles respectively. Individual points in Fig. 15 are outliers. We normalized all metric values to compare the distributions of metrics in the same scale (Normalized Metric Values in Fig. 15).

The distributions of individual metrics in Fig. 15 show different tendencies of defect-proneness. For example, the metric M18 shows a high degree of discrimination between buggy and clean instances. If we classify instances as buggy

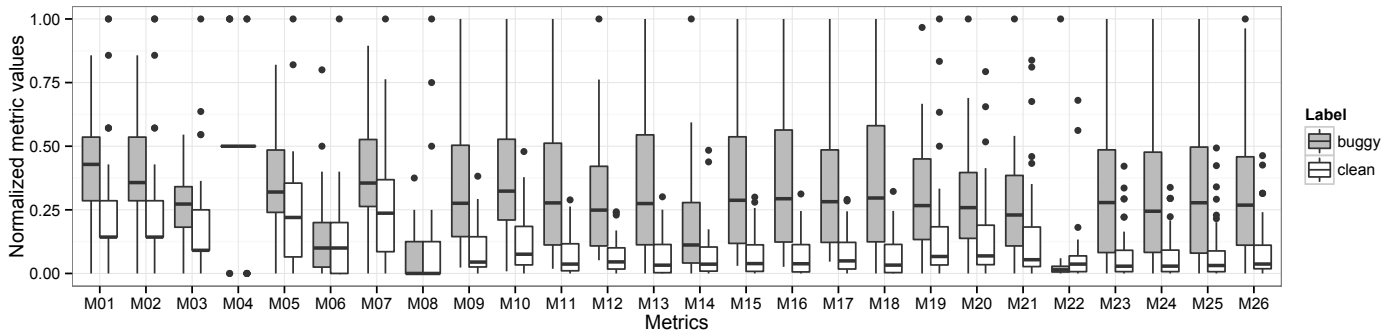

Fig. 15: Distributions of metric values of buggy and clean instances in Safe.

when the normalized metric value of M18 is greater than 0.12, about 75% of buggy and clean instances can be classified correctly. The higher metric value of M18 implies more defect-proneness. However, metrics such as M06 have less discriminative power between buggy and clean instances as shown in Fig. 15. Thus, M06 may be a less effective predictor for the Safe dataset when building a prediction model. The defect-proneness tendency of the metric, M22, is reversed; its higher metric values show less defect-proneness. In CLAMI, metrics such as M22 are automatically ignored since a majority of values in M22 are considered as violations by CLAMI.

The supervised models (WPDP) for the Safe dataset were built using all 26 metrics so that some metrics that do not have discriminative power in terms of defect-proneness can degrade the prediction performance of the models. The metrics, M06, M08, and M22, have relatively low discriminative power as there is little difference in the defect-proneness tendencies.

The CLAMI models were built using around six metrics on average of the Safe dataset since CLAMI applies metric selection to generate a training dataset. The most frequently selected six metrics from the Safe dataset are M13, M15, M16, M18, M23, and M25.

In Fig. 15, these six metrics show a clear tendency of defect-proneness in Safe. In other words, the CLAMI models can be constructed with metrics that have more discriminative power than using all 26 metrics that also include several metrics with relatively low defect-proneness tendencies. This can be a major reason CLAMI models could outperform supervised learning models in many project datasets as shown in Table II. There are previous studies where defect prediction performance can be improved further by using a small subset of selected metrics [33], [40], [62]. Our study also confirms these studies on the impact of metric selection.

We observe similar trends with other datasets except for Lucene after investigating the distributions of the metrics of each dataset. In the case of Lucene, the selected metrics by CLAMI do not follow the defect-proneness tendency of the typical defect prediction metrics. For this reason, CLA and CLAMI do not outperform the supervised learning models in f-measure and AUC as shown in Table II.

D. Threats to Validity

We carefully chose publicly available defect datasets such as NetGene and ReLink dataset groups that were generated using manually verified issue reports and links between issue reports and code changes respectively. However, NetGene datasets may have the quality issue of links between the manually verified issue reports and code changes since linking issue reports and code changes was still conducted automatically [44], [63]. In the case of ReLink, the issue reports were not manually verified although linking the issue reports to code changes were conducted manually [48]. Validating CLA/CLAMI with more reliable datasets generated with manual effort may be needed. However, to the best of our knowledge, these two defect dataset groups were generated with manual verification compared to other available datasets.

The CLA/CLAMI approaches are evaluated on defect datasets from seven open-source projects. Thus, the generalization of our results may be an external threat. However, we observe the potential of CLA/CLAMI as they can work well on datasets that follow the design rationale of defect prediction metrics. However, CLA/CLAMI may not work on the datasets that do not follow the defect-proneness tendency of typical defect prediction metrics. Since this can be a limitation of CLA/CLAMI, we have a plan to conduct additional experiments on various defect datasets as future work.

We implement THD and EXP approaches as real experts know the correct knowledge about threshold values and cluster labels. However, our implementation may not be the same as the real experts work so there could be a bias to compare CLA and CLAMI to THD and EXP.

VII. Conclusion

Enabling defect prediction for new projects or projects with limited historical information has long been a challenging issue. To address this limitation, we proposed CLA/CLAMI, which can build a prediction model on unlabeled datasets in an automated manner. In our empirical study on seven open-source projects, CLA/CLAMI models led to better or comparable results to typical defect prediction models and other baseline approaches in most projects in terms of recall, f-measure, and/or AUC. In addition, we observed CLAMI models work well on project datasets just using a small subset of selected metrics that follow the typical defect-proneness tendencies of the datasets. This result implies that in practice CLA and CLAMI have the potential for defect prediction on unlabeled datasets without need for manual effort. To evaluate the applicability of our approach in industry, we plan to apply CLA and CLAMI to proprietary software projects.

REFERENCES

[1] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, pp. 2–13, January 2007.

[2] E. Engström, P. Runeson, and G. Wikstrand, "An empirical evaluation of regression testing based on fix-cache recommendations," in *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, April 2010, pp. 75–78.

[3] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 452–461. [Online]. Available: http://doi.acm.org/10.1145/1134285.1134349

[4] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *Software Engineering, IEEE Transactions on*, vol. 22, no. 12, pp. 886–894, Dec 1996.

[5] T. Ostrand, E. Weyuker, and R. Bell, "Predicting the location and number of faults in large software systems," *Software Engineering, IEEE Transactions on*, vol. 31, no. 4, pp. 340–355, April 2005.

[6] P. Tomaszewski, H. Grahn, and L. Lundberg, "A method for an accurate early prediction of faults in modified classes," in *Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference on*, Sept 2006, pp. 487–496.

[7] A. Bacchelli, M. D'Ambros, and M. Lanza, "Are popular classes more defect prone?" in *Proceedings of the 13th International Conference on Fundamental Approaches to Software Engineering*, ser. FASE'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 59–73.

[8] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Softw. Eng.*, vol. 22, pp. 751–761, October 1996.

[9] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: Examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 4–14. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025119

[10] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[11] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09, 2009, pp. 78–88.

[12] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, "Predicting faults from cached history," in *Proceedings of the 29th international conference on Software Engineering*, ser. ICSE '07, 2007, pp. 489–498.

[13] T. Lee, J. Nam, D. Han, S. Kim, and I. P. Hoh, "Micro interaction metrics for defect prediction," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2011.

[14] M. Li, H. Zhang, R. Wu, and Z.-H. Zhou, "Sample-based software defect prediction with active and semi-supervised learning," *Automated Software Engineering*, vol. 19, no. 2, pp. 201–230, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10515-011-0092-1

[15] H. Lu and B. Cukic, "An adaptive approach with active learning in software fault prediction," in *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '12. New York, NY, USA: ACM, 2012, pp. 79–88. [Online]. Available: http://doi.acm.org/10.1145/2365324.2365335

[16] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proceedings of the 30th international conference on Software engineering*, ser. ICSE '08, 2008, pp. 181–190.

[17] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th international conference on Software engineering*, ser. ICSE '05, 2005, pp. 284–292.

[18] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. New York, NY, USA: ACM, 2009, pp. 91–100.

[19] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 531–540.

[20] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proceedings of the 2013 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 432–441.

[21] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Inf. Softw. Technol.*, vol. 54, no. 3, pp. 248–256, Mar. 2012.

[22] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings of the 2013 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 382–391.

[23] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Softw. Eng.*, vol. 14, pp. 540–578, October 2009.

[24] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter languagereuse," in *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*. New York, NY, USA: ACM, 2008, pp. 19–24.

[25] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the "imprecision" of cross-project defect prediction," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. New York, NY, USA: ACM, 2012, pp. 61:1–61:11.

[26] G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Multi-objective cross-project defect prediction," in *Software Testing, Verification and Validation, 2013 IEEE Sixth International Conference on*, March 2013, pp. 252–261.

[27] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'union fait la force," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, Feb 2014, pp. 164–173.

[28] C. Catal, U. Sevim, and B. Diri, "Clustering and metrics thresholds based software fault prediction of unlabeled program modules," in *Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference on*, April 2009, pp. 199–204.

[29] S. Zhong, T. Khoshgoftaar, and N. Seliya, "Unsupervised learning for expert-based software quality estimation," in *High Assurance Systems Engineering, 2004. Proceedings. Eighth IEEE International Symposium on*, March 2004, pp. 149–155.

[30] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 182–191. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597078

[31] B. Turhan, "On the dataset shift problem in software engineering prediction models," *Empirical Software Engineering*, vol. 17, no. 1-2, pp. 62–74, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10664-011-9182-8

[32] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. on Knowl. and Data Eng.*, vol. 22, pp. 1345–1359, October 2010.

[33] S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim, "Reducing features to improve code change-based bug prediction," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 552–569, 2013.

[34] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *Software Engineering, IEEE Transactions on*, vol. 34, no. 4, pp. 485–496, 2008.

[35] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 356–370, 2011.

[36] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *Software Engineering, IEEE Transactions on*, vol. 38, no. 6, pp. 1276–1304, Nov 2012.

[37] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. New York, NY, USA: ACM, 2014, pp. 172–181.

[38] M. Harman, S. Islam, Y. Jia, L. Minku, F. Sarro, and K. Srivisut, "Less is more: Temporal fault predictive performance over multiple hadoop releases," in *Search-Based Software Engineering*, ser. Lecture Notes in Computer Science, C. Le Goues and S. Yoo, Eds. Springer International Publishing, 2014, vol. 8636, pp. 240–246.

[39] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artif. Intell.*, vol. 97, no. 1-2, pp. 245–271, Dec. 1997. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(97)00063-5

[40] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: An investigation on feature selection techniques," *Softw. Pract. Exper.*, vol. 41, no. 5, pp. 579–606, Apr. 2011. [Online]. Available: http://dx.doi.org/10.1002/spe.1043

[41] H. Wang, T. M. Khoshgoftaar, and N. Seliya, "How many software metrics should be selected for defect prediction?" in *FLAIRS Conference*, R. C. Murray and P. M. McCarthy, Eds. AAAI Press, 2011. [Online]. Available: http://dblp.uni-trier.de/db/conf/flairs/flairs2011.htmlWangKS11

[42] 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," *Software Engineering, IEEE Transactions on*, vol. 39, no. 8, pp. 1040–1053, 2013.

[43] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 481–490. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985859

[44] K. Herzig, S. Just, A. Rau, and A. Zeller, "Predicting defects using change genealogies," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, Nov 2013, pp. 118–127.

[45] C.-L. Chang, "Finding prototypes for nearest neighbor classifiers," *Computers, IEEE Transactions on*, vol. C-23, no. 11, pp. 1179–1184, Nov 1974.

[46] E. Kocaguneli, T. Menzies, A. Bener, and J. Keung, "Exploiting the essential assumptions of analogy-based effort estimation," *Software Engineering, IEEE Transactions on*, vol. 38, no. 2, pp. 425–438, March 2012.

[47] Y. F. Li, M. Xie, and T. N. Goh, "A study of project selection and feature weighting for analogy based software cost estimation," *J. Syst. Softw.*, vol. 82, no. 2, pp. 241–252, Feb. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2008.06.001

[48] R. Wu, H. Zhang, S. Kim, and S. Cheung, "Relink: Recovering links between bugs and changes," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2011.

[49] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 392–401. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486840

[50] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: Bias in bug-fix datasets," in *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 121–130. [Online]. Available: http://doi.acm.org/10.1145/1595696.1595716

[51] Understand 2.0. [Online]. Available: http://www.scitools.com/products/

[52] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering*. New York, NY, USA: ACM, 2011, pp. 1–10.

[53] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, pp. 13–23.

[54] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, "High-impact defects: a study of breakage and surprise defects," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. New York, NY, USA: ACM, 2011, pp. 300–310.

[55] J. Nam, "Survey on software defect prediction," Department of Compter Science and Engineerning, The Hong Kong University of Science and Technology, Tech. Rep., 2014.

[56] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proc. of the 37th Int'l Conf. on Software Engineering (ICSE)*, ser. ICSE '15, 2015, pp. 789–800.

[57] R. Marinescu, "Detection strategies: metrics-based rules for detecting design flaws," in *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*, Sept 2004, pp. 350–359.

[58] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 414–423. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568320

[59] E. Giger, M. D'Ambros, M. Pinzger, and H. C. Gall, "Method-level bug prediction," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. New York, NY, USA: ACM, 2012, pp. 171–180.

[60] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248547.1248548

[61] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'"," *Software Engineering, IEEE Transactions on*, vol. 33, no. 9, pp. 637–640, Sept 2007.

[62] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, no. 0, pp. 170 – 190, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584914002523

[63] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, ser. PROMISE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 9–. [Online]. Available: http://dx.doi.org/10.1109/PROMISE.2007.10