Received March 21, 2020, accepted May 26, 2020, date of publication June 10, 2020, date of current version June 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3001440

CDS: A Cross–Version Software Defect Prediction Model With Data Selection

JIE ZHANG[1,2], **(Graduate Student Member, IEEE), JIAJING WU**[1,2], **(Senior Member, IEEE), CHUAN CHEN**[1,2], **(Member, IEEE), ZIBIN ZHENG**[1,2], **(Senior Member, IEEE), AND MICHAEL R. LYU**[3], **(Fellow, IEEE)**
[1]School of Computer and Data Science, Sun Yat-sen University, Guangzhou 510006, China
[2]National Engineering Research Center of Digital Life, Sun Yat-sen University, Guangzhou 510006, China
[3]Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong

Corresponding author: Jiajing Wu (wujiajing@mail.sysu.edu.cn)

This work was supported in part by the National Key Research and Development Program under Grant 2016YFB1000101, and in part by the National Natural Science Foundation of China under Grant 61973325, Grant 61503420, and Grant 61722214.

ABSTRACT Over the past decade, a large number of software defect prediction approaches have been proposed to identify the defect-prone modules by mining software repositories. Recently, a novel scenario called Cross–Version Defect Prediction (CVDP) begins to draw increasing research interests, as it is more reasonable and applicable in practice to adopt the labeled defect data of previous versions to predict defects in the current version of the same project. As a software project often has multiple previous versions, CVDP on this kind of projects will face the following two critical but seldom reported issues, namely, data distribution difference and class overlapping. In this paper, we address these two issues by solving a version selection problem via a Cross–version model with Data Selection (CDS). The proposed CDS is a novel framework which treats the defect prediction of existing and new files in different ways. For the existing files, we propose a novel Clustering–based Multi–Version Classifier (CMVC), which can automatically select the training data from the most relevant and noise-free versions by assigning them higher weights than the others. We proposed a Weighted Sampling Model (WSM) for the new files which have never appeared in previous version by incorporating the outputs of CMVC. We evaluate the proposed CDS model on 28 versions across 8 software projects, and the experimental results demonstrate that CDS outperforms three baseline methods and a state-of-the-art approach in terms of three prevalent performance indicators.

INDEX TERMS Software defect prediction, data selection, cross–version defect prediction.

I. INTRODUCTION

Defects in a software system may cause improper behaviors and even lead to great financial loss and critical safety accidents. Traditionally, techniques such as testing and code reviews are adopted to identify and correct defects in software systems. However, it can be excessively time-consuming and infeasible to test all of the components in the increasingly large–scale and complex software system in modern days. Considering the limited resource available, defect prediction, which aims to automatically identify the most defect-prone modules, has attracted profound attentions in the area of software engineering.

Defect prediction is often formulated as a supervised binary classification problem. The prediction models are trained with labeled defect data extracted from software repositories. Based on the source of training and testing datasets, defect prediction models proposed in previous work [1]–[7] can be categorized into two main categories: Within–Project Defect Prediction (WPDP) and Cross–Project Defect Prediction (CPDP). In WPDP models, both the training and test sets are collected from the same software project. While for CPDP models, training data from several different software projects are utilized for defect prediction task in a particular project.

Compared with WPDP, CPDP approaches can overcome the problem of data insufficiency but often yields relatively worse prediction performance due to the difference between the data distributions of the source and the target projects. On the other hand, most WPDP models proposed in existing literature are evaluated based on cross-validation [2], [3], [8]–[10]. In experiments of these

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

studies, labeled data from a specific version of a software project are randomly divided into training and test sets, and thus this kind of models are referred to as Inner–Version Defect Prediction (IVDP). In practice, however, a particular software is usually developed in successive versions, and it is impractical to use the data from the current or future versions to predict the defects in the upcoming version. To this end, recent studies [11]–[15] proposed to train the classification model by utilizing the labeled modules of previous versions of a project, and then predict defect modules in its subsequent version, i.e., Cross–Version Defect Prediction (CVDP).

Though several methods have been proposed for CVDP, most of them consider only one prior version for model training. However, it's common for a software project to have multiple previous versions and CVDP on this kind of projects faces the following two critical but seldom reported issues, namely, data distribution difference and class overlapping, which will be discussed in the next section. To address the two issues, in this paper, we propose a Cross-version defect prediction model with Data Selection (CDS), which treats the defect prediction of existing and new files in distinct ways. Experiments on 28 versions across 8 software projects demonstrate the superiority of CDS compared over baseline methods. The main contributions of the paper can be summarized as follows:

1) We put forward and validate two critical but seldom mentioned issues in CVDP, i.e., the variation of data distribution difference and the class overlapping problem caused by same–name files' repeated appearances in more than one versions. We conducted extensive statistical studies to demonstrate the existence of these two issues and whether they affect the prediction performance of CVDP models.
2) We proposed a novel cross–version defect prediction model called CDS to solve the above-mentioned issues by using two different prediction strategies for existing and new files. In particular, the CDS predicts the labels of existing files by minimizing the proposed objective function, while the defect labels of new files are predicted in an weighted sampling based manner.
3) We evaluated the proposed CDS on 28 versions across 8 software projects with three prevalent performance indicators, namely, F-measure, g-mean and Balance. Experimental results demonstrate that CDS achieves significant performance improvement compared with three traditional baseline methods and a state-of-the-art CVDP method.

II. BACKGROUND AND MOTIVATION

Defect prediction in software engineering is often formulated as a supervised binary classification problem. Based on the source of training and test data, defect prediction techniques can be classified into two main categories: within-project defect prediction (WPDP) and cross-project defect prediction (CPDP).

A. CROSS–PROJECT DEFECT PREDICTION

The scenario of CPDP has been proposed to address the problem of data insufficiency, which WPDP often suffers from, by utilizing training data from other projects. Various CPDP approaches have been proposed in previous work. Zimmermann *et al.* investigated the feasibility of cross-project defect prediction in [5]. Nam *et al.* [1] adopted the transfer learning approach to make data distribution of project similar. Later in [16], Peters *et al.* considered the privacy problem in CPDP. Zhang *et al.* [6] proposed an unsupervised model to handle the homogeneity between software projects. While in [17], Wu *et al.* applying an unified semi-supervised approach to deal with the insufficiency of historical data for both cross project and within project scenarios.

Compared with WPDP, the performance of CPDP is worse due to the difference between the data distributions of the source and the target projects. Although several approaches based on transfer learning [1], [18] have been adopted to tackle this problem, the prediction accuracy of the CPDP models is still relatively low because of some other factors, such as dissimilar project contexts, different development settings and heterogeneous set of metrics, etc. [19].

B. INNER–VERSION DEFECT PREDICTION

In previous studies, the issue of WPDP has been extensively investigated. Early studies extract software metrics from the perspective of complexity [20], [21], Object-Oriented [22], software dependency network [2], software process [23] and so on [8], [24]–[26]. In recent years, various machine learning techniques have been employed to enhance the prediction performance. In [3], Xiao *et al.* proposed a dictionary-based prediction method. Lu *et al.* combined semi-supervised learning with dimension reduction to achieve better prediction performance [27]. Yang el al. applying a ranking model to optimize the efficiency [28]. Besides, some methods are designed to address problems such as data noise [29] and class imbalance [4], [30]–[32].

Under the WPDP scenario, most prediction models are designed and tested based on the method of cross-validation, which divides the data collected from one particular version of a software project randomly into training and test sets, i.e., Inner–Version Defect Prediction (IVDP). In IVDP, data in the training and test sets have identical or similar data distributions and share the same bug pattern. In this way, IVDP can usually achieve decent prediction results.

However, the disadvantage of IVDP being often ignored is its inapplicability for industry use. The rationale behind the IVDP models is the assumption that the training and test sets are randomly selected from all collected data and should share the identical data distribution. Yet many practical cases deviate from this assumption, in that the defect prediction models can only be trained using the labeled data collected from previous versions, and then applied to predict defect-prone modules in the current or upcoming versions. However, datasets collected from different versions do not necessarily share the same data distribution, which is contradictory to the assumption of cross-validation.

C. CROSS–VERSION DEFECT PREDICTION

As discussed above, IVDP may not be applicable for practice use due to its dependency on cross-validation. On the other hand, despite various approaches to assimilate different distributions, the CPDP still cannot ensure solid performance. Therefore, in this paper, we pay special attention to the scenario of Cross–Version Defect Prediction (CVDP), in which only the data from previous versions within a particular software project are used for training and the trained model is tested on the current version of this software project. In the CVDP scenario, factors like project context, development settings, software architectures are often similar or identical in different versions of the same project, resulting in greater similarities among the data distributions and bug patterns of different versions, compared with those of different projects.

Shukla *et al.* formulated the CVDP problem as a multi-objective optimization problem and proposed a multi-object logistic regression model [13]. In [12], the method of kernel PCA was adopted to improve the prediction performance of CVDP. Yang *et al.* applied ridge regression model to deal with the multicollinearity problems [14]. Lu *et al.* noticed that the data distribution can differ from version to version, and integrated the method of active learning and dimensionality reduction to address this issue [11]. However, they did not investigated whether the distribution difference can significantly impact the prediction performance.

However, although a software project commonly has multiple previous versions, existing CVDP models usually use the collected data from only one prior version for model training and do not consider the issues of data distribution difference variance and class overlapping in the CVDP scenario with multiple previous versions:

1) DATA DISTRIBUTION DIFFERENCE

A software project usually has multiple versions, and the correlation and similarity between the data from each prior version and the current one can vary to a large extent. Besides, existing research has demonstrated that the defects collected by mining software repositories usually contain a certain amount of noise due to mislabeling [29], [33], and multiple previous versions may suffer from different extents of data noise. Most machine learning classifiers are designed under the assumption that the training and test data share the same data distribution, and the distribution difference between the training and test data can significantly affect the classification performance. Previous studies on CVDP usually train the prediction models using the data from one particular version (commonly the latest one) or the merged data of all prior versions. However, to improve the prediction performance of CVDP, it is more reasonable to choose the prior version whose data distribution is most similar to the current version.

2) CLASS OVERLAPPING

In software development, software projects update in a dynamic process and the prior version evolves into the subsequent versions by adding, deleting or modifying some source code files. Two versions of the same software project usually contain a large number of files with the same names. In current defect prediction approaches, several features of software modules are extracted from a variety of aspects such as complexity, object-orientation, dependency relationships, etc.. In most cases, there exists little difference between the files with the same name and thus the extracted features are mostly identical [34]. However, such files may be labeled oppositely in different versions for the following two reasons: (1) Although the corresponding changes for files with the same name may be tiny, they can induce or remove software defects. (2) Mislabeling may exist during the collection of defect information because mistakes are ubiquitous in human activities. Therefore, datasets collected from different versions of the same software project may contain instances with similar features but opposite labels, which can further cause the issue of class overlapping for the classification problem and result in degradation of the predictive performance.

Motivated by these two critical but rarely discussed issues for CVDP, we propose a novel cross-version defect prediction model with data selection to address the following three research questions:

RQ1 **For the same software project, is there a significant difference between the data distributions of various versions? And does this difference affect the performance of defect prediction models?**

RQ2 **Are there a large number of same–name files which are similar in terms of the extracted features but labeled oppositely?**

RQ3 **If the above two problems exist, how can one propose an effective method to improve the overall prediction performance of CVDP?**

III. PRELIMINARY STUDIES

To answer RQ1 and RQ2, we conducted empirical studies on the MORPH dataset [35] collected by Jurecko and Madeyski, which has been widely used in previous defect prediction studies [4], [7], [12], [36], [37]. The MORPH dataset contains 28 versions of 8 software systems, each instance in the dataset represents a source code file and consist of 20 object-oriented features and the defect label. The features are extracted from source code by using the CKJM tools [38]. The detailed description of these features can be found in [35] and a brief explanation of them is listed below.

- **WMC**: weighted methods per class
- **DIT**: depth of Inheritance Tree
- **NOC**: number of children

- **CBO**: coupling between object classes
- **RFC**: response for a class
- **LCOM**: lack of cohesion in methods
- **LCOM3**: lack of cohesion in methods, different from LCOM
- **NPM**: number of public methods
- **DAM**: data access metric
- **MOA**: weighted methods per class
- **MFA**: measure of functional abstraction
- **CAM**: cohesion among methods of class
- **IC**: inheritance coupling
- **CBM**: coupling between methods
- **AMC**: average method complexity
- **AC**: afferent couplings
- **EC**: efferent couplings
- **Max(CC)**: max value of McCabe's cyclomatic complexity
- **Avg(CC)**: average value of McCabe's cyclomatic complexity
- **LOC**: lines of code

A. STUDY ON RQ1

The empirical study on RQ1 consists of two parts. Firstly, we investigate the data distribution difference between different versions of the same software project by analyzing the statistical properties. Secondly, we compare prediction performance of classification models built on data of different versions to validate the data distribution difference can significantly affect the prediction performance.

In most software projects, a large number of source code files in a later version are inherited from previous versions with tiny changes, which may make the features of different versions very similar. However, the defect labels of a source code file can vary from version to version. Therefore, on the task of investigating the data distribution difference, we focus on comparing the ratio of defective source code files of different software versions.

In Table 1, the total number of files and the ratio of defective source code files in each version are compared. We can observe that different versions of the same project are obviously different in terms of both scale and defect ratio. For example, in the first project 'camel', only 3.8% files are defective in version 1.0, while the defect ratio increases dramatically to 35.5% in version 1.2 and then decreases to 16.6% in the next version. Thus, we can conclude that the obvious difference of data distributions between versions of the same software project does exist.

$$F-measure = \frac{2 * Precision * Recall}{Precision + Recall}, \quad (1)$$

$$g-mean = \sqrt{(\frac{TP}{TP+FN}) * (\frac{TN}{TN+FP})}, \quad (2)$$

$$Balance = 1 - \sqrt{\frac{(0-pf)^2 + (1-Recall)^2}{2}}. \quad (3)$$

TABLE 1. File numbers and defect ratios of the MORPH dataset.

Software Projects	Version	File Num	Defect Ratio
camel	1.0	339	3.8%
	1.2	608	35.5%
	1.4	872	16.6%
	1.6	965	19.5%
jedit	3.2	272	33.1%
	4.0	306	24.5%
	4.1	312	25.3%
	4.2	367	13.1%
log4j	1.0	135	25.2%
	1.1	109	33.9%
	1.2	205	92.2%
poi	1.5	237	59.5%
	2.0	314	11.8%
	2.5	385	64.4%
	3.0	442	63.6%
synapse	1.0	157	10.2%
	1.1	222	27.0%
	1.2	256	33.6%
velocity	1.4	196	75.0%
	1.5	214	66.4%
	1.6	229	34.1%
xalan	2.4	723	15.2%
	2.5	803	48.2%
	2.6	885	46.4%
xerces	init	162	47.5%
	1.2	440	16.1%
	1.3	453	15.2%
	1.4	588	63.7%

TABLE 2. Basic metrics for defect prediction.

	Actual	
Classified as	Defective	Non-defective
Defective	True Positive (TP)	False Positive(FP)
Non-defective	False Negative (FN)	True Negative (TN)
$Precision$	$Recall$	pf
$\frac{TP}{TP+FP}$	$\frac{TP}{TP+FN}$	$\frac{FP}{FP+TN}$

Next, we conduct defect prediction experiments on these software projects using three popular classification algorithms, i.e., Random Forest [39], Logistic Regression [40] and Naive Bayes [41].

The prediction performance is measured with three indicators, namely, *F-measure*, *g-mean* and *Balance*, which have been widely adopted in existing studies on defect prediction [12], [42], [43]. These three indicators are defined in (1), (2) and (3) according to the basic metrics given in TABLE 2, respectively.

In the experiments, the classification models are built using the Weka tool [44] with default parameters. As shown in TABLE 3, the performance of a particular defect prediction model can vary a lot due to the difference of training data from various versions. Again, taking the 'camel' project as an example, all the three classification algorithms perform poorly when the models are trained on version 1.0 which contain only 3.8% defective files. However, when using version 1.2 as the training set, the prediction performance

TABLE 3. Prediction performance of three benchmark classification models on each cross–version pair.

Software Project	Version Pair		Random Forest			Logistic Regression			Naive Bayes		
	Prior	Current	F-measure	g-mean	Balance	F-measure	g-mean	Balance	F-measure	g-mean	Balance
camel	1.0	1.6	0.071	0.193	0.319	0.106	0.240	0.334	0.248	0.402	0.412
	1.2	1.6	**0.417**	**0.638**	**0.634**	**0.296**	**0.487**	**0.479**	**0.338**	**0.522**	**0.506**
	1.4	1.6	0.342	0.504	0.485	0.189	0.337	0.375	0.301	0.475	0.465
	1.0+1.2+1.4	1.6	0.370	0.542	0.520	0.240	0.396	0.408	0.306	0.482	0.472
jedit	3.2	4.2	0.435	0.752	0.751	0.468	0.771	0.771	0.422	0.713	**0.708**
	4.0	4.2	0.476	0.729	0.714	0.528	0.727	0.698	**0.500**	**0.728**	0.707
	4.1	4.2	0.504	0.775	0.772	**0.567**	**0.790**	**0.777**	0.473	0.693	0.666
	3.2+4.0+4.1	4.2	**0.481**	**0.783**	**0.783**	0.504	0.746	0.731	0.475	0.712	0.691
log4j	1.0	1.2	0.498	**0.559**	**0.527**	0.455	0.527	0.500	0.441	0.500	0.487
	1.1	1.2	0.470	0.504	0.496	**0.478**	**0.527**	**0.509**	**0.453**	**0.509**	**0.495**
	1.0+1.1	1.2	**0.506**	0.529	0.518	0.410	0.493	0.474	0.453	0.509	0.495
poi	1.5	3.0	**0.768**	0.641	0.629	**0.764**	0.607	0.591	**0.463**	0.533	**0.513**
	2.0	3.0	0.182	0.313	0.365	0.101	0.230	0.331	0.214	0.345	0.378
	2.5	3.0	0.713	0.617	0.615	0.745	0.585	0.573	0.462	**0.538**	0.511
	1.5+2.0+2.5	3.0	0.654	**0.649**	**0.642**	0.635	**0.622**	**0.618**	0.294	0.410	0.417
synapse	1.0	1.2	0.280	0.410	0.416	0.372	0.485	0.465	0.504	0.600	0.568
	1.1	1.2	**0.444**	**0.557**	**0.533**	0.430	**0.545**	**0.524**	**0.538**	**0.639**	**0.620**
	1.0+1.1	1.2	0.362	0.489	0.477	**0.439**	0.544	0.513	0.510	0.614	0.592
velocity	1.4	1.6	0.488	0.233	0.332	0.486	0.256	0.340	0.448	0.243	0.330
	1.5	1.6	**0.596**	**0.617**	**0.586**	**0.554**	**0.554**	**0.535**	0.493	**0.601**	**0.583**
	1.4+1.5	1.6	0.562	0.564	0.541	0.531	0.350	0.382	**0.526**	0.548	0.538
xalan	2.4	2.6	0.428	0.523	0.493	0.372	0.479	0.458	**0.534**	**0.605**	**0.565**
	2.5	2.6	**0.680**	**0.684**	**0.683**	**0.483**	**0.543**	**0.540**	0.425	0.520	0.502
	2.4+2.5	2.6	0.643	0.676	0.672	0.284	0.408	0.414	0.428	0.523	0.498
xerces	init	1.4	**0.456**	0.274	0.283	**0.553**	0.274	0.302	0.268	0.380	0.402
	1.2	1.4	0.149	0.258	0.334	0.045	0.151	0.309	0.214	0.342	0.378
	1.3	1.4	0.241	**0.371**	**0.390**	0.209	**0.342**	**0.375**	**0.301**	**0.418**	**0.419**
	init+1.2+1.3	1.4	0.234	0.352	0.386	0.070	0.191	0.319	0.225	0.352	0.383

improves significantly for all three algorithms under all indicators. Therefore, data distribution difference does affect the performance of defect prediction models on a large scale.

To select the training data under the scenario of CVDP, some previous studies prefer the latest version, or simply merge data from all prior versions as the training set. According to TABLES 1 and 3, using the latest version or the merged data as training data cannot ensure the best prediction performance. Therefore, a more reasonable and dedicated method is required to select the training data in CVDP.

Based on the experimental results discussed above, we conclude the answer to RQ1 as follows. There does exist a significant difference between the data distributions of various versions for the same software project, and such a difference has a considerable influence on the prediction performance of CVDP models. Therefore, choosing training data with less noise and similar data distribution to the test set is of great importance for building CVDP models.

B. STUDY ON RQ2

In CVDP, changes of a source code file between two different versions may induce or fix defects, but it can hardly be reflected in the extracted features. Therefore, for different versions, there may exist some instances with similar features but opposite defect labels.

In TABLE 4, we count and record the number of files with the same file names between every pair of versions in the column 'Same File', and then calculate and record the number and percentage of oppositely labeled files in the column 'Conflict Label'.

Results listed in TABLE 4 indicate that almost every version pair contains a large number of same–name files, and a considerable percentage of these files are labeled oppositely. Moreover, to investigate whether the cross–version changes of these files can be reflected in the features, we calculate the average similarity of the corresponding instances using the cosine similarity [45]. Surprisingly, we find that the average cosine similarities of these same–name files all equal 1, which indicates that the cross-version changes of a file can hardly be reflected by the features.

Thus for RQ2, we can conclude that there are a great number of same–name source code files from different versions of a software project. Cross-version changes of these files can hardly be detected by currently used features. However, these files may be labeled differently due to the defect inducing and fixing nature of these changes. Therefore, CVDP dataset does contain a large number of instances that have similar features but opposite labels. Such a phenomenon can lead to the issue of class overlapping, which will largely degrade the performance of defect prediction models.

IV. METHOD

With the first two research questions answered, it is critical to propose a CVDP approach to solve the two issues listed in Section II-C, namely, data distribution difference and class overlapping. In this work, we address these two issues by solving a version selection problem because the first issue

TABLE 4. Class overlapping of each cross–version pair.

Software Project	Version Pair		Same File	Conflict Label	Average Similarity
	Prior	Current			
camel	1.0	1.2	262	78 (29.8%)	1.0
	1.0	1.4	260	54 (20.8%)	1.0
	1.0	1.6	257	65 (25.3%)	1.0
	1.2	1.4	569	133 (23.4%)	1.0
	1.2	1.6	565	158 (28.0%)	1.0
	1.4	1.6	857	151 (17.6%)	1.0
jedit	3.2	4.0	265	58 (21.9%)	1.0
	3.2	4.1	253	50 (19.8%)	1.0
	3.2	4.2	238	55 (23.1%)	1.0
	4.0	4.1	291	52 (17.9%)	1.0
	4.0	4.2	272	54 (19.9%)	1.0
	4.1	4.2	291	61 (21.0%)	1.0
log4j	1.0	1.1	98	20 (20.4%)	1.0
	1.0	1.2	107	66 (61.7%)	1.0
	1.1	1.2	103	64 (62.1%)	1.0
poi	1.5	2.0	224	126 (56.2%)	1.0
	1.5	2.5	224	41 (18.3%)	1.0
	1.5	3.0	222	76 (34.2%)	1.0
	2.0	2.5	314	202 (64.3%)	1.0
	2.0	3.0	311	178 (57.2%)	1.0
	2.5	3.0	382	122 (31.9%)	1.0
synapse	1.0	1.1	152	43 (28.3%)	1.0
	1.0	1.2	149	42 (28.2%)	1.0
	1.1	1.2	219	71 (32.4%)	1.0
velocity	1.4	1.5	155	72 (46.5%)	1.0
	1.4	1.6	153	102 (66.7%)	1.0
	1.5	1.6	209	89 (42.6%)	1.0
xalan	2.4	2.5	689	296 (43.0%)	1.0
	2.4	2.6	655	225 (34.4%)	1.0
	2.5	2.6	766	292 (38.1%)	1.0
xerces	init	1.2	116	38 (32.8%)	1.0
	init	1.3	114	67 (58.8%)	1.0
	init	1.4	70	51 (72.9%)	1.0
	1.2	1.3	433	96 (22.2%)	1.0
	1.2	1.4	323	217 (67.2%)	1.0
	1.3	1.4	328	176 (53.7%)	1.0

ask for selecting the prior version with the minimum data distribution difference as the training set, while the second one can be solved by deciding which version to trust when facing the problem that the same-name files from different versions are labeled contradictorily.

Here we address the version selection problem by giving each version a proper weight in the learning process. We propose a Cross–version defect prediction model with Data Selection (CDS) to address the version selection problem by learning a proper weight for each version and predict the labels of the current version in the same learning process.

A. METHOD OVERVIEW

Suppose we have m labeled datasets of previous versions $X^1, X^2, \ldots, X^m$ and the current version dataset X. The i-th version contains n_i instances $X^i = \{x_1^i, x_2^i, \ldots, x_{n_i}^i\}$ labeled with $f^i = (f_1^i, f_2^i, \ldots, f_{n_i}^i)$ where f_j^i takes 1 if the corresponding file is defective, and -1 if it is non-defective. The target of CVDP is to predict the labels of the current version $f = (f_1, f_2, \ldots, f_n)$.

Fig. 1 provides an overview of the proposed CDS which consists of three main parts: Data Processor, Clustering-based Multi–Version Classifier (CMVC) and Weighted Sampling Model (WSM). The Data Processor splits X into existing files X^{exist} and new files X^{new}, with X^{exist} referring to the files that have appeared in any previous version and X^{new} denoting the files that have never appeared in any previous version. Further, to facilitate later calculation, Data Processor extends the label vectors, $f^1, f^2, \ldots, f^m$, to the same size by adding zeros, and calculates the indicator matrix I^i for each previous version, which is a diagonal matrix with $I_{kk}^i = 1$ if the file with index k appears in both the i-th and the current versions, and $I_{kk}^i = 0$ otherwise.

When predicting the labels, X^{exist} and X^{new} are treated differently. The reason why we design this novel strategy is that most machine learning or statistical algorithms are based on a fundamental assumption that the instances of a dataset should be independently sampled from the same distribution. This assumption may hold true for the new files. Yet for the instances from X^{exist}, as discussed in Section II, they are usually dependent on the same–name files in the training dataset.

Here, CMVC is the core module of CDS, and is designed to solve the version weighting problem and to predict the labels of X^{exist} by solving an optimization problem (the rationale and a detailed description of CMVC will be given in Section IV-B). Further, with the version weight problem solved, WSM is designed to predict the labels of X^{new} using a classic classification model trained on the data sampled from previous versions according to the outcome version weights of CMVC.

B. CLUSTERING–BASED MULTI–VERSION CLASSIFIER

In this work, we propose a novel Clustering–based Multi–Version Classifier (CMVC) to assign each prior version a proper weight and obtain the predicted labels of the existing files by minimizing an objective function.

The intuitive ideas of our designed objective function can be summarized as follow:

I1 **Files with similar features should be labeled similarly**, which is a basic assumption of machine learning methods.

I2 **Files in the current version may inherit defect patterns from previous versions.** When predicting the labels of files in the current version, labels of the same-name files in the previous versions should be considered. More specifically, the label of a file in the current version is more likely to be similar to its labels in the previous versions with higher weights (higher data relevance and less noise).

I3 **Previous versions with higher data relevance should be given higher weights.** As discussed in II-C, the data distribution of different versions can vary a lot, giving higher weights to more relevant versions can improve the prediction performance.

I4 **Previous versions with less data noise should be given higher weights.** As discussed in II-C, defect prediction datasets may contain mislabeling noises. Therefore, we should assign larger weights to the versions with less noises.

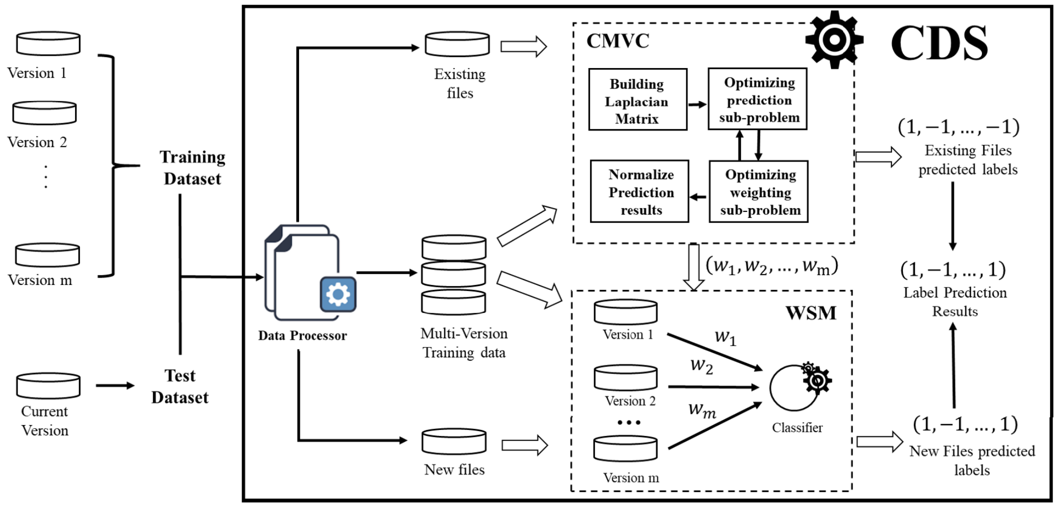

FIGURE 1. Overview of proposed CDS model.

1) OBJECTIVE FUNCTION

Based on the intuitive ideas above, we design the objective function of CMVC as fellows:

Firstly, to implement the first idea I1, we propose to use a clustering method to identify the instances with similar features and then assign these instances with identical or similar labels. Among various clustering techniques, here we employ spectral clustering [46] because it does not require an explicit model of data distribution and its effectiveness for handling data with a complex and unknown space shape. Spectral clustering is based on the graph Laplacian matrix of the data, which is defined as $L = D - A$, where A is the adjacency matrix of the similarity graph with A_{ij} representing the similarity between two instances, and D is a diagonal matrix with $D_{ii} = \sum_{j=1}^{n} A_{ij}$ (n is the number of instances). As an unsupervised learning algorithm, spectral clustering aims to minimize the following objective function:

$$\min J_f = f^T L f = \frac{1}{2}\sum_{i,j=1}^{n} A_{ij}(f_i - f_j)^2 \qquad (4)$$

where f is the label vector of the instances. Apparently, to minimize this objective function, similar instances should be given the same or similar labels.

Secondly, we consider the implementation of I2. To facilitate understanding, we first assume that the weight of each version w_i has been determined, which will be discussed later. The second idea can be implemented as:

$$\min \quad J_f = \sum_{i=1}^{m} w_i \frac{1}{c_i} \left\| I^i(f - f^i) \right\|_2^2 \qquad (5)$$

where f^i and w_i are the label vector and weight of the i-th version, respectively, c_i refers to the number of same-name files between current versions and the i-th version, and I_i refers to the indicator matrix of the i-th version.

In (5), $\left\| I^i(f - f^i) \right\|_2^2$ represents the label vector difference between the current version and the i-th previous version. We use $\frac{1}{c_i}$ to eliminate the influence of the number of same-name files on the results, and I_i to exclude the files that do not exist in both versions. By minimizing this function, the label of a file in the current version will be more similar to its labels in previous versions with higher weights.

After implementing these two ideas about label prediction, we next consider the implements of the third idea I3 and the fourth idea I4 about version weights. Again, to facilitate understanding, we can also assume that the labels of the current version are fixed when discussing the version weights.

We also use the label vector difference to measure the data relevance and implement I3. As investigated in Section III-B, the same-name files from different versions usually have similar features, while their labels can distinct a lot. Thus, the label difference of these files can be used to reflect the data relevance from different versions. Therefore, the third idea is implemented as:

$$\begin{aligned} \min \quad & J_w = \sum_{i=1}^{m} w_i \frac{1}{c_i} \left\| I^i(f - f^i) \right\|_2^2 + \lambda \left\| w \right\|_2^2 \\ s.t. \quad & w^T 1 = 1, \quad w_i \geq 0 \end{aligned} \qquad (6)$$

where, $w = (w_1, w_2, \ldots, w_m)$ is the version weights vector and λ is a non-negative parameter. This formula looks similar to (5), but the variable of it is changed to w. The first term in (6) represents the label vector difference between the current version and the i-th previous version, which can reflect the data relevance. The second term is used for the regularization of version weights. By minimizing this function, previous versions that are less different from the current version in terms of label vectors are considered to have higher data relevance and will be given higher weights.

Finally, we consider data noise in each previous version to decide the version weights, which refers to

the fourth idea I4. Since the labels of previous versions are known, we can measure the degree of data noise in each previous version based on the principle of (4). The prior version with a larger ${f^i}^T L^i f^i$ is more likely to have the instances with similar features but distinctive labels, indicating that this version may contain more noise and should be given a lower weight. Thus, considering the factor of noise, we can estimate the weights of previous versions by minimizing the following function:

$$\begin{aligned} \min \quad & J_w = \sum_{i=1}^{m} w_i \frac{1}{n_i^2} {f^i}^T L^i f^i + \lambda \|w\|_2^2 \\ s.t. \quad & w^T 1 = 1, \quad w_i \geq 0 \end{aligned} \tag{7}$$

where n_i, f^i and L^i denote the number of instances, the label vector and graph Laplacian matrix of the i-th version, respectively. w is the vector of weights for each previous version. $1/n_i^2$ is multiplied to avoid the effect of instances size and the second term is used for regularization of version weights. Clearly, by minimizing (7), the versions with less noise (a lower value of $\frac{1}{n_i^2} {f^i}^T L i f^i$ value) will be given a higher weight.

Taking all the aforementioned ideas into consideration, the complete objective function is designed as:

$$\begin{aligned} \min \quad & J_{w,f} = f^T L f + \lambda_1 \sum_{i=1}^{m} w_i \frac{1}{n_i^2} {f^i}^T L^i f^i \\ & + \lambda_2 \sum_{i=1}^{m} w_i \frac{1}{c_i} \left\| I^i (f - f^i) \right\|_2^2 + \lambda_3 \|w\|_2^2 \\ s.t. \quad & w^T 1 = 1, w_i \geq 0 \end{aligned} \tag{8}$$

where $\lambda_1, \lambda_2, \lambda_3$ are non-negative parameters to control each term.

The first term of this objective function is based on the spectral clustering of the current version. By minimizing this term, similar instances in the current version will be classified with the same label. ${f^i}^T L^i f^i$ in the second term estimates the degree of noise in the i-th version, and the versions with less noise will be assigned with larger weights in the learning process. The third term considers the relevance between previous versions and the current version, the irrelevant data can be removed by lowering the weights of corresponding versions. Besides, it makes the label vector of the current version similar to version with higher data relevance. Finally, the fourth term is a regularization of version weights.

For better understanding of (8), we can analyze the two variable, the label vector of current version f and the version weights w, individually. Fixing f, versions with lower $\frac{\lambda_1}{n_i^2} {f^i}^T L^i f^i + \frac{\lambda_2}{c_i} \left\| I^i (f - f^i) \right\|_2^2$ values will be given higher weights, in which the first term estimates the data noise and the second term measure the data relevance. While if we fix w, the label vectors of the current version is determined by minimizing $f^T L f + \frac{\lambda_2}{c_i} \left\| I^i (f - f^i) \right\|_2^2$, in which the first term ensure files with similar features have similar labels, and the second term makes the label of a file in the current version similar to its labels in the previous versions with higher weights.

Overall, by minimizing (8), data with high relevance and low noise are chosen from all previous versions by assigning higher weights to the corresponding versions, and the labels of the current version are predicted by considering feature similarity and defect patterns inherited from previous versions with large weights. In this way, the two issues discussed in Section III can be addressed during the learning process.

2) PROBLEM SOLVING

The optimization of the objective function can be decomposed into two sub-problems, a version-weighting sub-problem for version weight vector w and a label prediction sub-problem for instance label vector f. That is to say, we can minimize the objective function by alternatively fixing one of the vectors, and solving the other.

Fixing w, $J(f)$ is a non-negative convex function and can be solved as:

$$f = (L + \lambda_2 \sum_{i=0}^{m} \frac{1}{c_i} w_i I^i)^{-1} \lambda_2 \sum_{i=0}^{m} \frac{1}{c_i} w_i I^i f^i \tag{9}$$

In case $L + \lambda_2 \sum_{i=0}^{m} \frac{1}{c_i} w_i I^i$ is not an invertible matrix, we calculate the pseudo-inverse of matrix, instead denoted as $(L + \lambda_2 \sum_{i=0}^{m} \frac{1}{c_i} w_i I^i)^{\dagger}$, which can be obtained using the Least Square Method [47].

On the other hand, by fixing f, the version-weighting sub-problem is equivalently reduced to the following optimization problem:

$$\begin{aligned} \min \quad & v^T w + \lambda_3 \|w\|_2^2 \\ s.t. \quad & w^T 1 = 1, w_i \geq 0 \end{aligned} \tag{10}$$

where $v = (v_1, v_2, \ldots, v_m)$, with v_i being given by $\lambda_1 \frac{1}{n_i^2} {f^i}^T L^i f^i + \lambda_2 \frac{1}{c_i} \left\| I^i (f - f^i) \right\|_2^2$. This problem can be solved using the method proposed in [48] as follows:

$$w_i = \begin{cases} \dfrac{\theta - v_i}{2\lambda_3} & i \leq K \\ 0 & i > K, \end{cases} \tag{11}$$

where

$$\theta = \frac{2\lambda_3 + \sum_{i=1}^{K} v_i}{K}, \tag{12}$$

and

$$K = arg \max_i (\theta - v_i > 0). \tag{13}$$

Due to the limited space, the detailed proof procedure is not given here. Interested readers can refer to [48] for more details about the derivation.

The algorithm of minimizing the objective function of CMVC is summarized in Algorithm 1. As the two sub-problems have closed-form solutions, and the objective function decreases in each iteration. Therefore, it is guaranteed to converge to an optimal solution.

Algorithm 1 Algorithm for the Problem of Minimizing the Objective Function

Input: m previous versions data $X^1, X^2, \ldots, X^m$ and existing files data in the current version X^{exist}, the extended label vector of previous versions $f^1, f^2, \ldots, f^m$, and parameters $\lambda_1, \lambda_2, \lambda_3$.

Output: the binary label prediction vector f' and version-weight vector w

```
1: Initialize $w$ and $f$
2: Building Laplacian matrix $L^1, L^2, \ldots, L^m$ and $L$
3: repeat
4:     Optimize $f$ according to (9)
5:     Optimize $w$ according to (11)
6: until convergence
7: Convert $f$ to the binary label vector $f'$
```

C. WEIGHTED SAMPLING MODEL

To predict the labels of the files that have never appeared in previous versions, namely X^{new}, we propose the Weighted Sampling Model (WSM) to build the training set and then train the classification model via classic algorithms.

Since CMVC already addressed the two critical issues by giving each version a proper weight, we can directly utilize the outcome weight vector w to build the training set. WSM samples data from each previous version according to the version's corresponding weight. Since versions with a higher data relevance and less noise are assigned with higher weights, a larger amount of instances will be sampled from these versions. In such a manner, we can build a more reasonable training set than simply using the data of all previous versions or choosing only one version.

Specifically, we set the size of training set as the total number of instances in all previous versions, which is $N = \sum_{i=1}^{m} n_i$. For the i-th version, $N \cdot w_i$ instances will be randomly sampled with replacement. In this way, when building the training set, WSM relies more on previous versions with higher weights, which are more relevant to the current version and contain less noise according to CMVC.

After that, the prediction model will be trained on the sampled dataset with a prevalent classification algorithm, such as Random Forest, Logistic Regression, Naive Bayes, etc.. In this work, we choose Random Forest to train the classification model.

V. EXPERIMENT

In this section, we present the experimental setup and results. The experiments aim to investigate the following two questions:

Q1 How does the proposed method perform compared with other CVDP approaches?

Q2 Whether both CMVC and WSM achieve performance improvement by considering version weights?

To answer these two questions, the experiments are designed and conducted as follow.

A. EXPERIMENTAL DESIGN

1) BENCHMARK DATASET

In this work, we conducted experiments on 28 versions across 8 software projects from MORPH dataset [35]. We choose this dataset for two main reasons: firstly, it has been widely used in previous studies [4], [7], [12], [36], [37]. Secondly, it is the only dataset we found that contains multiple previous versions for each software project.

As mentioned in Section III-A, referring to TABLE 1, both the scale and distribution of data differ drastically between different versions of the same software project. Moreover, results in TABLE 4 indicates that the subsequent versions of a software project contain a large number of files inherited from previous versions, and these files across versions can be very similar in feature but are assigned different labels between versions. Detailed discussions can be found in Section III-B.

2) PERFORMANCE INDICATOR

The performance of our proposed method is evaluated with three indicators, namely *F-measure*, *g-mean* and *Balance*, which are used in our preliminary studies and also widely adopted in many previous defect prediction studies [12], [42], [43]. Due to the limitation of space, only these three indicators are reported in this manuscript, and results in terms of precision and recall, along with source codes are provided in additional materials [49].

To verify whether the prediction performances of two models have a significant difference, we use two statistical methods, i.e., the Wilcoxon signed-rank test [50] and Cliff's delta [51]. The Wilcoxon signed-rank test is a typical non-parametric test which can be used to check whether the performance difference between two compared methods is statistically significant. In addition, in this work, we use Cliff's delta as the effect size to quantify the difference. In this work, if the p value of the Wilcoxon test is lower than 0.05 and the d value of the Cliff's Delta is higher than 0.146, we reject the null hypothesis that the performance of two compared method have no significant difference with a confidence level of 95% [52].

3) METHODS FOR COMPARISONS

For the purpose of comparisons, we consider three baseline methods and a state-of-the-art CVDP approach. For the baseline methods, we adopt three classic and commonly adopted classification algorithms, namely Random Forest, Logistic Regression and Naive Bayes, which are widely used [11], [29], [43], [53]–[55] and compared [1], [3], [6], [27], [56] in the area of defect prediction.

For the state-of-the-art approach, Xu *et al.* proposed a two-phase CVDP framework by combining Hybrid Active Learning and Kernel PCA (HALKP) [12]. HALKP selects the most informative and representative files from current versions to query their labels, and then merges them into the training dataset to enhance the prediction performance. Experimental results in [12] demonstrated that HALKP

TABLE 5. The detailed results of CDS and the baseline methods.

Software	F-measure					g-Mean					Balance				
Projects	RF	LR	NB	HALKP	CDS	RF	LR	NB	HALKP	CDS	RF	LR	NB	HALKP	CDS
camel	0.370	0.240	0.306	0.429	**0.516**	0.542	0.396	0.482	0.583	**0.711**	0.520	0.408	0.472	0.552	**0.703**
jedit	0.481	0.504	0.475	**0.564**	0.526	0.783	0.746	0.712	0.708	**0.784**	0.783	0.731	0.691	0.699	**0.779**
log4j	0.506	0.410	0.453	0.613	**0.719**	0.529	0.493	0.509	0.622	**0.636**	0.518	0.474	0.495	0.598	**0.634**
poi	0.654	0.635	0.294	0.675	**0.768**	0.649	0.622	0.410	0.683	**0.705**	0.642	0.618	0.417	0.667	**0.703**
synapse	0.362	0.439	0.510	0.458	**0.538**	0.489	0.544	0.614	0.568	**0.642**	0.477	0.513	0.592	0.544	**0.638**
velocity	0.562	0.531	0.526	0.527	**0.610**	0.564	0.350	0.548	0.494	**0.697**	0.541	0.382	0.538	0.485	**0.693**
xalan	0.428	0.284	0.428	0.610	**0.709**	0.523	0.408	0.523	0.636	**0.692**	0.493	0.414	0.498	0.636	**0.682**
xerces	0.234	0.070	0.225	0.402	**0.538**	0.352	0.191	0.352	0.468	**0.509**	0.386	0.319	0.383	0.466	**0.509**
AVG	0.450	0.389	0.402	0.535	**0.616**	0.554	0.469	0.519	0.595	**0.672**	0.545	0.482	0.511	0.581	**0.668**
p-value	0.012	0.012	0.012	0.017		0.012	0.012	0.012	0.327		0.012	0.012	0.012	0.327	
cliff'd	0.719	0.781	0.953	0.359		0.563	0.719	0.672	0.578		0.563	0.719	0.813	0.438	

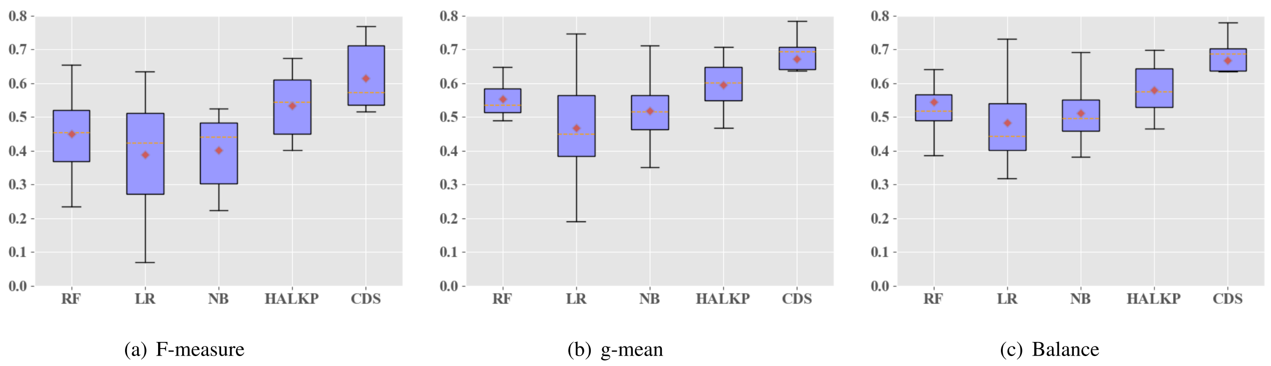

FIGURE 2. Box plots of baseline methods, HALKP and CDS on three indicators.

outperforms other CVDP approaches, and therefore we compare our proposed method with HALKP to evaluate its effectiveness.

In the experiments, both the proposed method and the compared methods are trained on the merged data of all previous versions to predict the defects in the last version. All the compared methods do not take the two critical issues we pointed out in II-C into consideration. Thus, by comparison with these methods, we can find out whether the proposed model can effectively overcome these issues.

B. EXPERIMENTAL RESULTS

1) ANSWERING Q1

We compare CDS with three baseline methods and one state-of-the-art CVDP approach on 8 software projects. The detailed experimental results are presented in TABLE 5, in which RF, LR and NB refer to the classification algorithm Random Forest, Logistic Regression and Naive Bayes, respectively, and AVER denotes the average value of each column. For each software project, the training set is the data in all previous versions, and the current version is taken as the test set. The entries highlighted in bold face represents the method that outperforms others.

As we can see from TABLE 5, CDS achieves the best performance both on average and almost on every single software project in terms of all the three indicators. The box plots given in FIGURE. 2 demonstrate the superiority of CDS more clearly.

Compared with the baseline methods, CDS obtains significant performance improvement which can be reflected by the fact that all the p values are below 0.05 and all the d values are much higher than 0.146. More specifically, in terms of *F-measure*, the improvement of CDS is between 36.9% and 58.3% over the baseline methods; in terms of *g-mean*, CDS is superior to baseline methods on all evaluated software projects with the average improvement varying from 21.3% to 43.2%; in terms of *Balance*, CDS also outperforms the baseline methods by 22.6% to 38.6% on average.

On the other hand, compared with the state-of-the-art method HALKP, CDS also achieves better performance in most cases. Although the p value is below 0.05 only on F-measure, the d values are all above the threshold. In particular, the average improvements are 15.1%, 13.0% and 15.0% in terms of *F-measure*, *g-mean* and *Balance*, respectively. Besides, as discussed before, HALKP requires defective information about the current version while our

TABLE 6. Prediction performance of existing files.

Software Projects	F-measure				G-mean				Balance			
	RF	LR	NB	CMVC	RF	LR	NB	CMVC	RF	LR	NB	CMVC
camel	0.407	0.247	0.315	**0.535**	0.567	0.401	0.489	**0.728**	0.540	0.412	0.477	**0.723**
jedit	0.500	0.524	0.500	**0.558**	0.780	0.753	0.726	**0.782**	**0.780**	0.739	0.708	0.772
log4j	0.519	0.430	0.426	**0.854**	0.518	0.523	0.489	**0.675**	0.513	0.486	0.479	**0.667**
poi	0.659	0.671	0.303	**0.792**	0.637	0.670	0.414	**0.698**	0.632	0.655	0.421	**0.694**
synapse	0.370	0.467	0.550	**0.559**	0.496	0.567	0.647	**0.659**	0.482	0.533	0.626	**0.658**
velocity	0.528	0.498	0.475	**0.567**	0.563	0.331	0.531	**0.675**	0.543	0.371	0.525	**0.666**
xalan	0.600	0.325	0.459	**0.676**	0.646	0.442	0.548	**0.655**	0.640	0.436	0.519	**0.645**
xerces	0.187	0.065	0.163	**0.627**	0.312	0.182	0.295	**0.560**	0.365	0.317	0.356	**0.560**
AVG	0.471	0.403	0.399	**0.646**	0.565	0.484	0.517	**0.679**	0.562	0.494	0.514	**0.673**
p-value	0.012	0.012	0.012		0.012	0.012	0.012		0.025	0.012	0.012	
cliff'd	0.719	0.844	0.969		0.656	0.656	0.781		0.688	0.719	0.781	

method does not. Therefore, we also find the performance improvement of CDS over HALKP satisfactory.

Therefore, we can conclude that the proposed CDS method can achieve better prediction results than the compared methods in the cross-version defect prediction scenario, meanwhile indicating that the CDS can effectively address the issues mentioned in Section II-C.

2) ANSWERING Q2

To further investigate the effects of the two core modules of our model, namely CMVC and WSM, we compare the performance of CMVC and WSM with baseline methods on the prediction task of the existing files and new files respectively. For fairness consideration, HALKP is not compared here. Specifically, HALKP is based on the method of active learning, which needs to query labels of instances in the test set. When the size of test set is small, this property of active learning may cause a certain degree of unfair comparison. Therefore, the prediction performances of CMVC and WSM are only compared with three classic baseline methods.

TABLE 6 compares the performance of CMVC and baseline methods on existing files, while the comparison of WSM and baseline methods on new files are presented in TABLES 7, 9 and 8. Since WSM can build the classifier on the sampled data using different classification algorithms, we compare them with the original algorithms individually. In TABLE 7, 9 and 8, W-RF, W-LR, and W-NB denotes the WSM classifier built with Random Forest, Naive Bayes, and Logistic Regression, respectively.

From the results in TABLE 6, we can see that CMVC outperforms the baseline methods on existing files in terms of all three indicators, and the p values and d values indicate that the performance improvement is significant. Moreover, it should be noted that CMVC achieves better performance than the overall performance of CDS (Refer to results in TABLE 5). As the existing files take up the majority

TABLE 7. Prediction performance of new files–RF.

Software Projects	F-measure		G-mean		Balance	
	RF	W-RF	RF	W-RF	RF	W-RF
camel	0.057	**0.258**	0.219	**0.458**	0.322	**0.454**
jedit	0.400	**0.414**	0.804	**0.811**	0.799	**0.807**
log4j	0.492	**0.516**	0.537	**0.554**	0.518	**0.533**
poi	0.595	0.432	0.684	**0.557**	**0.665**	0.547
synapse	0.316	**0.364**	0.449	**0.494**	0.449	**0.489**
velocity	0.800	**0.828**	0.514	**0.629**	0.492	**0.592**
xalan	0.847	**0.942**	0.810	**0.942**	0.784	**0.938**
xerces	0.278	**0.423**	0.396	**0.496**	0.407	**0.481**
AVG	0.473	**0.522**	0.552	**0.618**	0.555	**0.605**
p-value	0.123		0.093		0.092	
cliff'd	0.125		0.250		0.250	

TABLE 8. Prediction performance of new files–LR.

Software Projects	F-measure		G-mean		Balance	
	LR	W-LR	LR	W-LR	LR	W-LR
camel	0.174	**0.240**	0.335	**0.408**	0.375	**0.416**
jedit	0.400	**0.522**	0.705	**0.856**	0.683	**0.856**
log4j	0.390	**0.504**	0.463	**0.546**	0.459	**0.525**
poi	0.340	**0.440**	0.424	**0.537**	0.424	**0.537**
synapse	0.250	**0.353**	0.384	**0.470**	0.401	**0.455**
velocity	0.828	0.828	0.629	0.629	0.592	0.592
xalan	0.000	**0.225**	0.000	**0.356**	0.293	**0.384**
xerces	0.076	**0.131**	0.199	**0.261**	0.321	**0.342**
AVG	0.307	**0.405**	0.392	**0.508**	0.444	**0.513**
p-value	0.018		0.018		0.018	
cliff'd	0.297		0.297		0.266	

proportion of the whole software file, the overall performance improvement should be attributed mainly to CMVC.

For the new files, all the three tables show that WSM can obtain better prediction performance. The average performance of all the three indicators is higher than the original classification algorithms, and the p values and d values are all satisfactory only except the comparison with RF in terms of F-measure. It indicates that the adoption of a weighted sampling strategy using the weight vector

TABLE 9. Prediction performance of new files–NB.

Software Projects	F-measure		G-mean		Balance	
	NB	W-NB	NB	W-NB	NB	W-NB
camel	0.214	**0.276**	0.401	**0.463**	0.414	**0.456**
jedit	0.333	**0.400**	0.616	**0.705**	0.588	**0.683**
log4j	0.480	**0.527**	0.528	**0.563**	0.511	**0.540**
poi	0.174	0.174	0.312	0.312	0.363	0.363
synapse	0.222	**0.545**	0.367	**0.635**	0.395	**0.609**
velocity	0.867	**0.880**	0.655	**0.852**	0.596	**0.852**
xalan	0.216	**0.653**	0.346	**0.693**	0.382	**0.654**
xerces	0.279	**0.299**	0.403	**0.419**	0.408	**0.417**
AVG	0.348	**0.469**	0.454	**0.580**	0.457	**0.572**
p-value	0.018		0.018		0.018	
cliff'd	0.391		0.484		0.516	

learned by CMVC can build a better training set for classical classification algorithms. **Thus, we can conclude that** both CMVC and WSM achieve performance improvement over the baseline methods.

In summary, the experimental results give positive answers to both questions listed at the beginning of this section. CDS outperforms all three baseline methods and the state-of-the-art approach in terms of all three indicators, signifying the fact that it has indeed solved the two critical issues discussed in Section II-C. Therefore we conclude that CDS can be superior and more applicable for cross-version defect prediction.

VI. CONCLUSION

In this paper, we discussed the advantages of cross-version defect prediction (CVDP) for practical use compared with within-project defect prediction and cross-project defect prediction. Then we pointed out two critical issues that may pose significantly threat to the performance of CVDP models but are seldom mentioned in previous studies, and conducted preliminary studies to verify the existence of these two issues and the way they affect the CVDP performance. To solve these issues and improve the prediction performance of CVDP, we proposed a novel cross-version defect prediction model with data seletion (CDS), where the defect labels of existing and new files are predicted in distinct ways. In particular, we designed a novel clustering–based multi–version classifier (CMVC) to predict the defect labels of the existing files by automatically selecting the training data from the most relevant and noisy-free versions, and employ a Weighted Sampling Model (WSD) for defect prediction of new files. We evaluated the new framework on 28 versions across 8 projects acquired from a public defect dataset. The experimental results demonstrated that CDS outperforms three classic classification algorithms and a state-of-the-art CVDP method.

Future extensions of our work include the application of CDS on software projects with more previous versions, improving its performance on class-imbalance data and exploring how to extend CDS to CPDP scenarios to solve the problem of insufficient data for newly-started software projects.

REFERENCES

[1] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proc. IEEE 35th Int. Conf. Softw. Eng. (ICSE)*, San Francisco, CA, USA, May 2013, pp. 382–391.
[2] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proc. 30th Int. Conf. Softw. Eng.*, Leipzig, Germany, May 2008, pp. 531–540.
[3] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proc. 36th Int. Conf. Softw. Eng. (ICSE)*, Hyderabad, India, May 2014, pp. 414–423.
[4] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, "MAHAKIL: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction," *IEEE Trans. Softw. Eng.*, vol. 44, no. 6, pp. 534–550, Jun. 2018.
[5] 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 *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf., ACM SIGSOFT Symp. Found. Softw. Eng.*, 2009, pp. 91–100.
[6] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *Proc. IEEE 38th Int. Conf. Softw. Eng. (ICSE)*, Austin, TX, USA, May 2016, pp. 309–320.
[7] S. Herbold, "Training data selection for cross-project defect prediction," in *Proc. 9th Int. Conf. Predictive Models Softw. Eng.*, Baltimore, MD, USA, Oct. 2013, p. 6.
[8] T. Lee, J. Nam, D. Han, S. Kim, and H. P. In, "Micro interaction metrics for defect prediction," in *Proc. 19th ACM SIGSOFT Symp., 13th Eur. Conf. Found. Softw. Eng. (SIGSOFT/FSE)*, Szeged, Hungary, 2011, pp. 311–321.
[9] Z. Sun, Q. Song, and X. Zhu, "Using coding-based ensemble learning to improve software defect prediction," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 6, pp. 1806–1817, Nov. 2012.
[10] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, Jul. 2008.
[11] H. Lu, E. Kocaguneli, and B. Cukic, "Defect prediction between software versions with active learning and dimensionality reduction," in *Proc. IEEE 25th Int. Symp. Softw. Rel. Eng.*, Naples, Italy, Nov. 2014, pp. 312–322.
[12] Z. Xu, J. Liu, X. Luo, and T. Zhang, "Cross-version defect prediction via hybrid active learning with kernel principal component analysis," in *Proc. IEEE 25th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Campobasso, Italy, May 2018, pp. 209–220.
[13] S. Shukla, T. Radhakrishnan, K. Muthukumaran, and L. B. M. Neti, "Multi-objective cross-version defect prediction," *Soft Comput.*, vol. 22, no. 6, pp. 1959–1980, Mar. 2018.
[14] X. Yang and W. Wen, "Ridge and lasso regression models for cross-version defect prediction," *IEEE Trans. Rel.*, vol. 67, no. 3, pp. 885–896, Sep. 2018.
[15] Y. Huang, X. Hu, N. Jia, X. Chen, Y. Xiong, and Z. Zheng, "Learning code context information to predict comment locations," *IEEE Trans. Rel.*, vol. 69, no. 1, pp. 88–105, Mar. 2020.
[16] F. Peters, T. Menzies, and L. Layman, "LACE2: Better privacy-preserving data sharing for cross project defect prediction," in *Proc. IEEE/ACM 37th Int. Conf. Softw. Eng.*, Amsterdam, The Netherlands, May 2015, pp. 801–811.
[17] F. Wu, X.-Y. Jing, Y. Sun, J. Sun, L. Huang, F. Cui, and Y. Sun, "Cross-project and within-project semisupervised software defect prediction: A unified approach," *IEEE Trans. Rel.*, vol. 67, no. 2, pp. 581–597, Jun. 2018.
[18] 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.
[19] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 874–896, Sep. 2018.
[20] M. H. Halstead, *Elements of Software Science* (Operating and Programming Systems Series). New York, NY, USA: Elsevier Science, 1977.
[21] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *IEEE Trans. Softw. Eng.*, vol. 22, no. 12, pp. 886–894, Dec. 1996.

[22] 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, no. 10, pp. 751–761, Oct. 1996.

[23] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, San Francisco, CA, USA, May 2013, pp. 432–441.

[24] S. E. S. Taba, F. Khomh, Y. Zou, A. E. Hassan, and M. Nagappan, "Predicting bugs using antipatterns," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Eindhoven, The Netherlands, Sep. 2013, pp. 270–279.

[25] Q. Xie, Z. Wen, J. Zhu, C. Gao, and Z. Zheng, "Detecting duplicate bug reports with convolutional neural networks," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2018, pp. 416–425.

[26] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "MalPat: Mining patterns of malicious and benign Android apps via permission-related APIs," *IEEE Trans. Rel.*, vol. 67, no. 1, pp. 355–369, Mar. 2018.

[27] H. Lu, B. Cukic, and M. Culp, "Software defect prediction using semi-supervised learning with dimension reduction," in *Proc. 27th IEEE/ACM Int. Conf. Automat. Softw. Eng. (ASE)*, Sep. 2012, pp. 314–317.

[28] X. Yang, K. Tang, and X. Yao, "A learning-to-rank approach to software defect prediction," *IEEE Trans. Rel.*, vol. 64, no. 1, pp. 234–246, Mar. 2015.

[29] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *Proc. IEEE 33rd Int. Conf. Softw. Eng. (ICSE)*, Honolulu, HI, USA, May 2011, pp. 481–490.

[30] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Trans. Rel.*, vol. 62, no. 2, pp. 434–443, Jun. 2013.

[31] M. Liu, L. Miao, and D. Zhang, "Two-stage cost-sensitive learning for software defect prediction," *IEEE Trans. Rel.*, vol. 63, no. 2, pp. 676–686, Jun. 2014.

[32] L. Pelayo and S. Dick, "Evaluating stratification alternatives to improve software defect prediction," *IEEE Trans. Rel.*, vol. 61, no. 2, pp. 516–525, Jun. 2012.

[33] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The impact of mislabelling on the performance and interpretation of defect prediction models," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, Florence, Italy, vol. 1, May 2015, pp. 812–823.

[34] A. V. Phan, M. Le Nguyen, and L. T. Bui, "Convolutional neural networks over control flow graphs for software defect prediction," in *Proc. IEEE 29th Int. Conf. Tools Artif. Intell. (ICTAI)*, Boston, MA, USA, Nov. 2017, pp. 45–52.

[35] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proc. 6th Int. Conf. Predictive Models Softw. Eng.*, Timişoara, Romania, Sep. 2010, p. 9.

[36] L. Chen, B. Fang, Z. Shang, and Y. Tangc, "Negative samples reduction in cross-company software defects prediction," *Inf. Softw. Technol.*, vol. 62, pp. 67–77, Jun. 2015.

[37] S. Amasaki, K. Kawata, and T. Yokogawa, "Improving cross-project defect prediction methods with data simplification," in *Proc. 41st Eur. Conf. Softw. Eng. Adv. Appl.*, Madeira, Portugal, Aug. 2015, pp. 96–103.

[38] D. Spinellis, "Tool writing: A forgotten art?" *IEEE Softw.*, vol. 22, no. 4, pp. 9–11, Jul. 2005.

[39] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[40] S. Le Cessie and J. C. Van Houwelingen, "Ridge estimators in logistic regression," *J. Roy. Stat. Soc., C, Appl. Statist.*, vol. no. 1, pp. 191–201, 1992.

[41] G. H. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in *Proc. 11th Conf. Uncertainty Artif. Intell.*, Montreal, QC, Canada, Aug. 1995, pp. 338–345.

[42] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," in *Proc. 11th Working Conf. Mining Softw. Repositories (MSR)*, Hyderabad, India, May 2014, pp. 182–191.

[43] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Trans. Softw. Eng.*, early access, Oct. 17, 2018, doi: 10.1109/TSE.2018.2876537.

[44] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explor. Newslett.*, vol. 11, no. 1, pp. 10–18, 2009.

[45] H. V. Nguyen and L. Bai, "Cosine similarity metric learning for face verification," in *Proc. Asian Conf. Comput. Vis.* Berlin, Germany: Springer, 2010, pp. 709–720.

[46] K. Kamvar, S. Sepandar, K. Klein, D. Dan, M. Manning, and C. Christopher, "Spectral learning," in *Proc. Int. Joint Conf. Artif. Intell.*, 2003, pp. 1–6.

[47] S. M. Stigler, "Gauss and the invention of least squares," *Ann. Statist.*, vol. 9, no. 3, pp. 465–474, May 1981.

[48] C. Chen, J. Xin, Y. Wang, L. Chen, and M. K. Ng, "A semisupervised classification approach for multidomain networks with domain selection," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 1, pp. 269–283, Jan. 2018.

[49] J. Zhang, J. Wu, C. Chen, Z. Zheng, and M. R. Lyu. (2019). *Supplementary Material*. [Online]. Available: https://zenodo.org/record/2573882#.XG1kZ-gzaHs

[50] F. Wilcoxon, S. Katti, and R. A. Wilcox, "Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test," *Sel. Tables Math. Statist.*, vol. 1, pp. 171–259, Jun. 1970.

[51] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff's delta calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.

[52] N. Cliff, *Ordinal Methods for Behavioral Data Analysis*. London, U.K.: Psychology Press, 2014.

[53] T. Yu, W. Wen, X. Han, and J. H. Hayes, "ConPredictor: Concurrency defect prediction in real-world applications," *IEEE Trans. Softw. Eng.*, vol. 45, no. 6, pp. 558–575, Jun. 2019.

[54] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proc. IEEE 38th Int. Conf. Softw. Eng. (ICSE)*, May 2016, pp. 297–308.

[55] H. Hata, O. Mizuno, and T. Kikuno, "Bug prediction based on fine-grained module histories," in *Proc. IEEE 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 200–210.

[56] J. Nam and S. Kim, "CLAMI: Defect prediction on unlabeled datasets (T)," in *Proc. 30th IEEE/ACM Int. Conf. Automat. Softw. Eng. (ASE)*, Nov. 2015, pp. 452–463.

JIE ZHANG (Graduate Student Member, IEEE) received the B.S. degree in software engineering from the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2017. He is currently pursuing the master's degree with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His main research interests include software engineering and complex networks.

JIAJING WU (Senior Member, IEEE) received the B.Eng. degree in communication engineering from Beijing Jiaotong University, Beijing, China, in 2010, and the Ph.D. degree from The Hong Kong Polytechnic University, Hong Kong, in 2014.

In 2015, she joined Sun Yat-sen University, Guangzhou, China, where she is currently an Associate Professor. Her research interests include network science and its applications in engineering networked systems, such as communication networks, power grids, and cyber-physical systems. She was a recipient of the Hong Kong Ph.D. Fellowship Scheme during her Ph.D. degree, Hong Kong, from 2010 to 2014. She serves as an Associate Editor for the IEEE Transactions on Circuits and Systems-II: Express Briefs.

CHUAN CHEN (Member, IEEE) received the B.S. degree from Sun Yat-sen University, Guangzhou, China, in 2012, and the Ph.D. degree from Hong Kong Baptist University, Hong Kong, in 2016. He is currently an Associate Research Fellow with the School of Data and Computer Science, Sun Yat-sen University. His main research interests include machine learning, numerical linear algebra, and numerical optimization.

ZIBIN ZHENG (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2010.

He is currently an Associate Professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His research interests include services computing, software engineering, and block chain.

Dr. Zheng received the Outstanding Thesis Award of CUHK, in 2012, the ACM SIGSOFT Distinguished Paper Award from ICSE2010, and the Best Student Paper Award from ICWS, in 2010.

MICHAEL R. LYU (Fellow, IEEE) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1981, the M.S. degree in computer engineering from the University of California, Santa Barbara, CA, USA, in 1985, and the Ph.D. degree in computer science from the University of California, Los Angeles, CA, in 1988.

He is currently a Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He is also the Director of the Video over Internet and Wireless (VIEW) Technologies Laboratory. His research interests include software reliability engineering, distributed systems, fault-tolerant computing, mobile networks, web technologies, multimedia information processing, and e-commerce systems.

Dr. Lyu is a fellow of ACM, AAAS, and Croucher Senior Research.