The Journal of Systems & Software 172 (2021) 110862

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

A comprehensive comparative study of clustering-based unsupervised defect prediction models

Zhou Xu [a,b], Li Li [c], Meng Yan [a,b,*], Jin Liu [d,**], Xiapu Luo [e], John Grundy [c], Yifeng Zhang [d], Xiaohong Zhang [a,b]

[a] Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education, China
[b] School of Big Data and Software Engineering, Chongqing University, Chongqing, China
[c] Faculty of Information Technology, Monash University, Australia
[d] School of Computer Science, Wuhan University, Wuhan, China
[e] Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China

ARTICLE INFO

Article history:
Received 3 March 2020
Received in revised form 6 October 2020
Accepted 7 November 2020
Available online 16 November 2020

Keywords:
Clustering-based unsupervised models
Empirical study
Data analytics for defect prediction

ABSTRACT

Software defect prediction recommends the most defect-prone software modules for optimization of the test resource allocation. The limitation of the extensively-studied supervised defect prediction methods is that they require labeled software modules which are not always available. An alternative solution is to apply clustering-based unsupervised models to the unlabeled defect data, called **C**lustering-based **U**nsupervised **D**efect **P**rediction (**CUDP**). However, there are few studies to explore the impacts of clustering-based models on defect prediction performance. In this work, we performed a large-scale empirical study on 40 unsupervised models to fill this gap. We chose an open-source dataset including 27 project versions with 3 types of features. The experimental results show that (1) different clustering-based models have significant performance differences and the performance of models in the instance-violation-score-based clustering family is obviously superior to that of models in hierarchy-based, density-based, grid-based, sequence-based, and hybrid-based clustering families; (2) the models in the instance-violation-score-based clustering family achieves competitive performance compared with typical supervised models; (3) the impacts of feature types on the performance of the models are related to the indicators used; and (4)the clustering-based unsupervised models do not always achieve better performance on defect data with the combination of the 3 types of features.

1. Introduction

The defects hidden in software modules threaten the security and decrease the reliability of the software product. Therefore, it is essential to fix the defective modules before delivering the product.

Defect fixing is a complex and time-consuming task, and limited testing resources are usually unaffordable for supporting thorough code reviews (Geremia and Tamburri, 2018). This requests a prioritization to better analyze the software product. In other words, developers and testers should reasonably allocate the limited resources to test the modules that have a high probability to contain defects. To seek for such prioritization, **S**oftware **D**efect **P**rediction (**SDP**) is proposed to identify the most defect-prone modules for priority inspection. The most active SDP methods are supervised models which first train a classifier on labeled modules and then use it to determine whether or not the unlabeled modules contain defects. However, the supervised SDP models need the labeled modules of historical data of the current project or external projects which are not always available.

In order to conduct defect prediction on unlabeled data, **U**nsupervised **D**efect **P**rediction (**UDP**) models are possible for this task. As UDP models do not need any labeled data, they have attracted many researchers' attention in recent years. There are 2 types of UDP models: **C**lustering-based **U**nsupervised **D**efect **P**rediction (**CUDP**) methods (such as the studies Zhong et al., 2004a; Bishnu and Bhattacherjee, 2012; Zhang et al., 2016) and **R**anking-based **U**nsupervised **D**efect **P**rediction (**RUDP**) methods (such as the studies Yang et al., 2016; Fu and Menzies, 2017; Yan et al., 2017; Huang et al., 2017). RUDP methods select one feature to rank modules based on the corresponding values. The rationale behind this type of method is based on the assumption that the feature values and the defect-proneness of the modules have a direct or inverse proportional relationship (Yang et al., 2016). However, such a relationship does not exist in

[*] Corresponding author at: School of Big Data and Software Engineering, Chongqing University, Chongqing, China.
[**] Corresponding author.
E-mail addresses: mengy@cqu.edu.cn (M. Yan), jinliu@whu.edu.cn (J. Liu).

https://doi.org/10.1016/j.jss.2020.110862
0164-1212/© 2020 Elsevier Inc. All rights reserved.

all features, which leads to inconsistent conclusions in previous studies. For example, Yang et al. (2016) found that RUDP methods performed significantly better than supervised models on change-level just-in-time defect data, but Yan et al. (2017) found that the conclusion in Yang et al. (2016) does not hold on a file-level benchmark dataset. Thus, more work is needed to investigate and verify the generalization of RUDP on different defect data. In addition, RUDP methods need a threshold (such as the proportion of the top-ranked modules) to divide the modules into two groups for calculating some performance indicators, such as F-measure. However, this threshold is not easy to be determined. Unlike RUDP methods, CUDP methods do not rely on the relationship between a specific feature and the defect label to rank the modules, thus avoiding the above contradictory conclusions. CUDP methods divide the modules into different groups based on a specific rule without relying on a threshold. In this work, we focused on CUDP methods and their performance on defect data with different feature sets.

The general process of CUDP methods consists of the following 2 steps: (1) leveraging a similarity metric to cluster unlabeled modules into different groups where the modules in the same group are more similar to each other compared with those in other groups. This step is based on the information found in the data that describes the relationships among the modules; (2) applying a specific strategy to annotate each group as defective or non-defective. In previous studies, researchers have applied some clustering-based methods to unlabeled defect data. For example, in early studies, researchers employed classic clustering methods like K-means algorithm (Zhong et al., 2004a) and self-organizing maps algorithm (Abaei et al., 2013) to group the modules. In more recent studies, researchers designed specific methods to cluster the modules, such as clustering and label method (Nam and Kim, 2015), and average clustering method (Yang and Qian, 2016).

1.1. Motivation

There are several limitations in existing CUDP approaches: (1) there are few studies conducting a systematic literature review towards CUDP articles; (2) all previous studies focus on using existing methods or developing new methods to cluster unlabeled modules for SDP, but few studies have explored the performance differences of various clustering-based methods for UDP; (3) previous studies have shown that different feature types have impacts on the SDP performance of supervised models (Moser et al., 2008; Zimmermann and Nagappan, 2008; Radjenović et al., 2013; Kaur et al., 2015), but to our best knowledge, there is no study explored the impacts of feature types on the SDP performance of the clustering-based methods (i.e., the CUDP performance); and (4) all previous studies evaluated the CUDP performance with traditional indicators that do not consider the inspecting efforts for modules, but no study has employed the more practical effort-aware indicators.

Motivated by these limitations, in this work we conducted a large-scale empirical study to analyze the performance differences of 40 clustering-based unsupervised models (as well as 6 supervised models for comparison) on a public benchmark dataset. This dataset consists of 14 projects with a total of 27 versions in which 3 kinds of features are collected for each project. We evaluated these methods with one traditional and 2 effort-aware indicators. The experimental results show that (1) there exist significant performance differences among these methods, and the hierarchy-based, density-based, grid-based, sequence-based, and hybrid-based clustering models perform significantly worse for CUDP task in most cases; (2) some clustering-based unsupervised models, such as the instance-violation-score-based clustering methods, can achieve even better performance than the typical supervised models; (3) the CUDP performance of the methods on different indicators is affected by the feature types of the defect data; (4) the supervised models usually perform better on defect data with multiple feature types, while the phenomenon does not conform to the clustering-based unsupervised models.

1.2. Contribution

The main contributions of this study include:

(1) We retrieved and analyzed existing SDP studies involving clustering methods from different perspectives, such as the used datasets, feature types, performance indicators, clustering methods, and labeling schemes. To the best of our knowledge, this is the first work to conduct such a detailed analysis for CUDP studies.
(2) We applied 40 clustering-based models from 9 clustering families to 27 project versions who have 3 types of features. In addition, we employed both traditional and effort-aware indicators to evaluate the performance of these methods. To our best knowledge, we were among the first to conduct such a wide-ranging empirical study for investigating the impacts of feature types on the CUDP performance and use both kinds of indicators for synthetically evaluating the CUDP performance.
(3) We designed and implemented an experimental framework which integrates 40 clustering-based unsupervised SDP models from multiple libraries. We further made the framework public available and encouraged our fellow researchers to integrate their state-of-the-art clustering models to this framework for further comparative studies.

The remainder of the paper is organized as follows: Section 2 introduces the studied 40 clustering-based unsupervised models and summaries the existing studies related to CUDP. Section 3 describes the design of our empirical study. Section 4 reports our experimental results. Section 5 discusses the implications from the experimental results and the potential validity threats. Section 6 presents different types of empirical studies in SDP domain. Section 7 concludes this paper and draws potential future directions.

2. Taxonomy and literature review

2.1. Taxonomy for clustering-based unsupervised models

As clustering-based unsupervised models identify defective software modules without requiring the participation of labeled modules, it is meaningful to seek models that can achieve similar or better performance than supervised models for defect prediction. We briefly introduced our studied 40 unsupervised models from 9 clustering families.

2.1.1. ***P**artition-**B**ased **C**lustering (**PBC**) family*

Given a dataset D with n instances (i.e., the software modules), a predefined cluster number k, and an *objection function* F, PBC methods first construct $k(k \leq n)$ partitions of the data where each partition represents a cluster. Note that 2 conditions need to be satisfied: (1) each cluster must contain at least one instance and each instance must belong to exactly one cluster. Then PBC methods utilize the iterative relocation technique to optimize the *object function* F by moving instances from one group to another (Han et al., 2011). The aim is to make the instances in the same cluster close to each other, whereas modules in distinct clusters are far apart. The *object function* F is usually defined as the distances between each instance to its center instance point.

The typical processing method followed by the PBC family is: first, it randomly selects k instances as the initial center points and assigns each remaining instance to a cluster whose center point is nearest to that instance. Then, it updates the center instance of each cluster and relocates the clusters of other instances. This process iterates until meeting a predetermined condition, such as the center points of the clusters remain unchanged.

In this work, we studied 13 methods in PBC family, including K-Means (Hartigan and Wong, 1979), **C**ascade K-**M**eans(**CM**) (Karegowda et al., 2012), Canopy (McCallum et al., 2000), X-Means (Pelleg et al., 2000), K-Medoids (Jin and Han, 2016), **P**artitioning **A**round **M**edoids (**PAM**) (Kaufman and Rousseeuw, 2009), **M**ini **B**atch K-**M**eans (**MBM**) (Béjar Alonso, 2013), **F**uzzy **C**-**M**eans (**FCM**) (Bezdek et al., 1984), **F**uzzy **C**-**S**hell (**FCS**) (Dave, 1990), **H**ard **C**-**M**eans (**HCM**) (MacQueen et al., 1967), K-Modes (Huang, 1997), **F**arthes**F**irst (**FF**) (Hochbaum and Shmoys, 1985), **Cl**ustering **LAR**ge **A**pplications (**CLARA**) (Kaufman and Rousseeuw, 2009). These methods are basically the variations of K-means.

2.1.2. *Hierarchy-Based Clustering (HBC) family*

HBC methods recursively create a hierarchical decomposition of the data. According to the direction of the decomposition, HBC methods can be classified as either agglomerative hierarchical clustering methods (i.e, bottom-up decomposition) or divisive hierarchical clustering methods (i.e., top-down decomposition). The former treats each instance as a separate cluster at the beginning and successively merges the closest cluster into a larger one, until all instances are merged into one cluster or a predefined condition meets. The latter treats all instances as an initial cluster at the beginning and then successively splits the cluster into smaller ones until each instance belongs to one cluster or a predefined condition meets. The condition can be the desired cluster number or the inconsistency coefficient (Xu et al., 2016b).

In this work, we studied 6 methods in HBC family, including **A**gglomerative **H**ierarchical **C**lustering (**AHC**) (Ding and He, 2002), **D**ivisive **A**nalysis **C**luster (**DAC**) (Ding and He, 2002), **RO**bust **C**lustering using lin**K**s (**ROCK**) (Guha et al., 2000), **L**earning **V**ector **Q**uantization (**LVQ**) (Kohonen, 1995), **C**lustering **U**sing **RE**presentatives (**CURE**) (Guha et al., 1998), **B**alanced **i**terative **r**educing and **c**lustering using **h**ierarchies (**Birch**) (Zhang et al., 1996).

2.1.3. *Density-Based Clustering (DBC) family*

Methods in PBC family usually divide instances based on distance information, and thus work well on finding clusters of spherical shape rather than arbitrary shape (Han et al., 2011). The methods in the DBC family alleviate this limitation by using the notion of data distribution density. Given a radius r and a density threshold p for each instance, if its spherical region (the circular region in a two-dimensional plane) with radius r contains at least p instances, then all these instances construct a cluster.

In this work, we studied 3 methods in DBC family, including **D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise (**DBSCAN**) (Ester et al.), **O**rdering **P**oints **T**o **I**dentify **C**lustering **S**tructure (**OPTICS**) (Ankerst et al., 1999), and **M**ean **S**hift (**MS**) (Cheng, 1995).

2.1.4. *Grid-Based Clustering (GBC) family*

The methods in GBC family are based on the space-driven concept, which quantizes the feature space into a finite number of grid cells. These cells are independent of the distribution of input instances and form a grid structure. Each instance falls into a grid cell, which means that the feature space of the grid cell contains that instance. All the clustering operations are carried out on the grid structure.

In this work, we studied one method in GBC family, i.e., **CL**ustering **I**n **QUE**st (**CLIQUE**) (Agrawal et al., 1998).

2.1.5. *Model-Based Clustering (MBC) family*

The methods in MBC family assume a model for each cluster and seek instances that can best match the model. They obtain the clusters by constructing the density function of the spatial distribution of the instances. The most frequently-assumed model is the probability model and the division is based on the form of probability. These lead to the unified probability distribution of instances within the same cluster.

In this work, we studied 7 methods in MBC family, including **N**eural-**G**as (**NG**) (Martinetz et al., 1991), **E**xpectation **M**aximization (**EM**) (Dempster et al., 1977), Cobweb (Fisher, 1987), **S**elf-**O**rganizing **M**ap (**SOM**) (Kohonen, 1998), **SOM** for **S**imple **C**lustering (**SOMSC**) (Novikov, 2018), **SYNC**hronized **SOM** (**SYNCSOM**) (Novikov and Benderskaya, 2014), on-line update method (i.e., **H**ard **C**ompetitive **L**earning (**HCL**)) (Fritzke, 1997).

2.1.6. *Graph-Theory-Based Clustering (GTBC) family*

The methods in GTBC family first construct a weighted graph where each node represents an instance and the weight of the edge denotes the similarity measure of its two nodes. Then, they divide the graph into several subgraphs. As the division process is usually based on the local dependencies of the graph, GTBC methods can maintain the local connectivity on the data.

In this work, we studied 2 methods in the GTBC family, including **A**ffinity **P**ropagation (**AP**) (Frey and Dueck, 2007) and **S**pectral **C**lustering (**SC**) (Ng et al., 2002).

2.1.7. *Sequence-Based Clustering (SBC) family*

The methods in SBC family use the feature vectors once or multiple times to generate compact and hyper-ellipsoidal clusters. Their performance usually depends on the order in which the vectors are presented to the methods (Kainulainen and Kainulainen, 2002).

In this work, we studied 2 methods in SBC family, including **B**asic **S**equential **A**lgorithmic **S**cheme (**BSAS**) and **M**odified **BSAS** (**MBSAS**) (Theodoridis and Koutroumbas, 2006). BASA considers each instance only once while MBSAS runs twice through each instance.

2.1.8. *Instance-Violation-Score-Based Clustering (IVSBC) family*

The notation of IVS is derived from previous studies (Nam and Kim, 2015; Yang and Qian, 2016) that designed a specific clustering criterion for software modules in the context of SDP. Here, we used a simple example to describe the calculation process of this criterion. Given a defect data with 5 software modules (i.e., M1 − M5) and 4 features (i.e., F1 − F4) in Fig. 1, we further defined an initial violation matrix V whose elements are all 0. First, for each feature, we calculate one statistic value as the cutoff threshold. Here, we assumed the statistic value as the median value (Nam and Kim, 2015). Thus, the threshold vector of the 4 features is [3, 3, 4, 4]. Then, for each module, if its ith feature value is larger than the corresponding threshold, the value of its corresponding position in matrix V changes to 1. For example, comparing module M1 with feature vector [4, 2, 2, 5] and the corresponding threshold vector [3, 3, 4, 4], the values of the first entry and the fourth entry in the first row of matrix V are changed to 1 as showed in Fig. 1 with gray background. This process was repeated for all modules. After obtaining the final violation matrix V, the sum of each row is treated as the IVS of the corresponding module. Note that the threshold vector is defined as the **M**edian value of the **F**eature (**MF**) in Nam and Kim (2015) and as the **H**alf **A**verage value of the **F**eature (**HAF**) in Yang and Qian (2016) as showed at the bottom of the left part in Fig. 1. From the figure, we could observe that different choices of the threshold vector will result in different IVSs which are used as the measurement to divide the modules into distinct clusters.

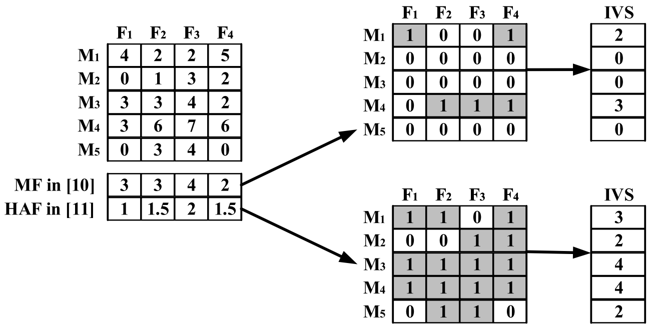


Fig. 1. An example of calculation process for IVS.

In this work, we studied 4 methods in IVSBC family, including **C**lustering and **LA**bel (**CLA**) (Nam and Kim, 2015), its improved version **C**lustering and **LA**bel with **M**etric selection and **I**nstance selection (**CLAMI**) (Nam and Kim, 2015), **A**verage **C**lustering (**AC**) (Yang and Qian, 2016), and **C**luster **E**nsembles (**CE**) (Yang et al., 2018).

2.1.9. *Hybrid Clustering (HC) family*

For methods that combine multiple clustering techniques, we classify them as in the HC family.

In this work, we studied 2 methods in HC family, including **H**ierarchical K-**M**eans **C**lustering (**HMC**) (Alboukadel, 2017a) and **H**ierarchical **C**lustering on **P**rincipal **C**omponents (**HCPC**) (Alboukadel, 2017b). Both of them combine hierarchical clustering method and K-means clustering method.

The concise descriptions for the 40 methods are presented in Table 1.

2.2. *Literature analysis*

In this section, we conducted a literature analysis of all existing studies related to CUDP.

2.2.1. *Search process*

To understand the research progress in CUDP, we conducted a search for the related articles that should satisfy the following 3 criteria: (1) the article applied clustering-based unsupervised learning methods to software defect data; (2) the article was written in English; (3) the full text of the article was available online. We used the combined terms "defect prediction"+"clustering", "fault prediction"+"clustering", "quality prediction"+"clustering" as well as "defect prediction"+"unsupervised", "quality prediction"+"unsupervised", "fault prediction"+"unsupervised" to search the related articles. As a result, we retrieved a total of 34 articles. Through carefully reading these papers, we found that 7 articles (Yang et al., 2006, 2016; Fu and Menzies, 2017; Yan et al., 2017; Huang et al., 2017, 2018; Chen et al., 2019) do not satisfy the first selection criterion. In addition, article (Gupta et al., 2012a) just simply introduced 4 clustering-based methods without conducting any qualitative and quantitative analysis on software defect data. Therefore, we removed the 8 articles and focused on the analysis of the remaining 26 articles as listed in the first column in Table 3. In addition, to verify the completeness of our search, we followed previous work (Zhou et al., 2018) to conduct a forward snowballing search. Note that we searched the articles published from 2000 because we found that the earliest articles using the clustering algorithm to analyze the defect data were published after that year. More specifically, we first searched and inspected the articles having cited the these articles through Google Scholar, then filtered out the unrelated articles. In this work, we followed the previous work (Zhou et al., 2018) to use Google Scholar as the main digital library, and also searched the articles in the ACM Digital Library, IEEE Xplore, Elsevier ScienceDirect, and SpringerLink to check if any articles have been omitted. We repeated this process on all the reserved articles. Table 2 reports the statistic information of the reserved papers based on the type and year.

2.2.2. *Existing unsupervised methods for SDP*

Table 3 summaries the information of the used datasets and performance indicators of the 26 selected articles including the published year, the number of used projects (Proj.), the corresponding development languages, the number and type of the corresponding features, the availability of the used dataset, the performance indicators, and the citations (Cit.). Note that the citations are counted from the Google Scholar on July 24, 2020.

From Table 3, we have the following observations: (1) In the articles published before 2015, the researchers conducted experiments on a small number of projects with fewer features and the corresponding feature type only consists of the code complexity metrics; (2) the used projects in these articles are mainly developed with Java, C++, and C; (3) In the articles published after 2012, most researchers employed the defect data that are available online as their studied corpora which is helpful for others to reproduce their experimental results. Note that the entries with gray background in the 7th column indicate that the authors had provided a link to the dataset, but the link to the web page fails at the moment; (4) the frequently-used performance indicators are classification accuracy, error, **F**alse **P**ositive **R**ate (**FPR**), and **F**ault **N**egative **R**ate (**FNR**) for articles published before 2015, while the recent articles usually used the comprehensive indicators, such as F-measure and AUC. However, no studies have investigated the performance of effort-aware indicators for their used clustering-based methods; (5) the citations of most studies are less than 50 and only five articles (Zhong et al., 2004a; Bishnu and Bhattacherjee, 2012; Zhang et al., 2016; Nam and Kim, 2015; Yuan et al., 2000) has more than 100 citations. This statistic indicates that, from the current situation, the CUDP topic has not attracted widespread attentions from the researchers.

Table 4 presents an overview of information about the unsupervised models used in these articles, including the specific clustering-based methods (the column 2–6), the number of the clusters (the column 7), and the used cluster labeling scheme (LS) (the column 8).

From Table 4, we have the following observations: (1) the methods in PBC and MBC families are frequently used for CUDP, but no methods in HBC, GBC, and HC families have been used. This inspires us to further investigate the impacts of these uninvestigated methods on CUDP; (2) half of the articles clustered the software modules into 2 groups, which is based on the fact that the defect data only contain 2 classes modules, i.e., the defective and non-defective modules. In addition, there were 6 articles that did not specify the cluster number in advance;

2.2.3. *Labeling schemes*

From the tables, we can find that there exist a total of 6 labeling schemes in previous studies (scheme 0 means that the authors did not mention how to label each cluster):

- Scheme 1 denotes the expert inspection based labeling strategy which invites experts to assign the label of each cluster;
- Scheme 2 denotes metric thresholds based labeling. This scheme defines 6 feature [Lines of Code, Cyclomatic Complexity, Unique Operator, Unique Operand, Total Operator, Total Operand] as [65, 10, 25, 40, 125, 70] as the threshold vector, then compares the vector with the feature of

Table 1
A summary of the studied unsupervised learning methods.

Fam.	Method	Brief Description	No.
PBC	K-means	A representative-based clustering by selecting the average values of the instances in the cluster as the centers	1
	K-medoids	Improving K-means by selecting the instances in the cluster as the centers	2
	CM	An improvement of K-means with automatic selection of K using the Calinski and Harabasz criterion	3
	X-means	An extension of K-means with efficiently searching the space of cluster locations and number	4
	MBM	A variant of K-means by using mini-batches to reduce the computation time	5
	PAM	An extension of K-means by finding a sequence of medoids that are centrally located in clusters	6
	FCM	The simplest fuzzy clustering algorithm which is a variant of K-means by allowing a instance to belong to more than one cluster	7
	FCS	A generalization of fuzzy clustering to shell like clusters, i.e. detecting clusters that lie in nonlinear subspaces	8
	HCM	An extension of basic K-means based on classical set theory requiring that a instance either does or does not belong to a cluster	9
	K-modes	An extension of K-means by replacing distances with dissimilarities and means with modes	10
	FF	A variant of K-means by replacing each cluster center in turn with the instance furthest from the existing cluster centers	11
	Canopy	Speeding up clustering operations on large datasets	12
	CLARA	Using sampling to handle large datasets with PAM	13
HBC	AHC	Building a larger cluster by merging two smaller clusters in a bottom-up fashion	14
	DAC	Splitting a cluster into two smaller ones in a top-down fashion	15
	Birch	Using clustering feature and the corresponding tree to improve clustering speed and scalability, especially on large datasets	16
	LVQ	Combining vector quantization and nearest-neighbor classification to update the cluster centers in an incremental manner	17
	CURE	Using instance variants from a constant number of well scattered instances after shrinking as the cluster representative for large datasets, even with non-spherical shapes and wide variances in size	18
	ROCK	Considering the number of common neighbors for a pair of instances during clustering	19
DBC	DBSCAN	Grouping together instances that have many nearby neighbors and marking outliers whose nearby neighbors are too far away	20
	OPTICS	Detecting meaningful clusters in spatial data of varying density	21
	MS	Iteratively shifting each instance in the dataset until the top of its kernel density estimation surface reaches a nearest peak	22
GBC	CLIQUE	Constructing static grids to perform a bottom-up subspace clustering and using a prior method to reduce the search space	23
MBC	NG	An artificial neural network for finding optimal data representations based on feature vectors	24
	EM	Iteratively performing an expectation (E) step, which creates a function for the expectation of the log-likelihood, and a maximization (M) step, which computes parameters by maximizing the log-likelihood	25
	Cobweb	Traversing a classification tree top-down starting from the root node to find the best inserting position of a new instance by calculating a category utility function	26
	SOM	A competitive learning network that uses a neighborhood function to preserve the topological properties of the input space	27
	SOMSC	An adaptation of SOM for cluster analysis in simple way by using amount of cluster that should be allocated as amount of neurons in the SOM	28
	SYNCSOM	A bio-inspired algorithm that is based on oscillatory network that uses SOM as the first layer	29
	HCL	A winner-take-all algorithm comprising methods where each input instance only determines the adaptation of one unit, i.e., the winner	30
GTBC	SC	Using the similarity matrix of the input data to construct a connected graph and treating the data clustering as a graph partitioning problem	31
	AP	Based on the concept of "message passing" between instances and selecting the real instances as the cluster centers for K-medoids	32
SBC	BSAS	Setting the cluster's representative as only a single vector and favoring the creation of compact clusters in which vectors are presented only once	33
	MBSAS	A modification to BSAS which runs twice through the instances	34
IVSBC	CLA	First clustering the modules and ranking the clusters based on the violation scores, then labeling the clusters in the top half as defective	35
	CLAMI	After the same process as CLA, then selecting the modules with metric selection and instance selection to build a supervised model	36
	ACL	Calculating the violation scores of all modules, then the modules whose scores are higher than a threshold are labeled as defective	37
	CE	Using clustering algorithm ACL on generated multiple data partitions and combining the multiple clusters into a single better one	38
HC	HMC	First computing the cluster centers with hierarchical clustering, then using the k-means with these centers as initial cluster centers	39
	HCPC	First performing hierarchical clustering on the selected principal components to obtain initial partitioning by cutting the hierarchical tree, then using k-means to refine the initial partition	40

Table 2
Statistic information of research papers published by type and year.

Year	2000	2001	2004	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2018	Total
Conference	2	1	1	1	1	1	1	1	0	0	2	1	1	2	2	17
Journal	0	1	1	0	0	0	0	2	1	2	0	2	0	0	0	9
Total	2	2	2	1	1	1	1	3	1	2	2	3	1	2	2	26

Table 3
A summary of previous studies related to CUDP.

Study	Year	Dataset characteristics					Performance indicators	Cit.
		Proj.	Language	Feature number	Feature type	Available?		
Yuan et al. (2000)	2000	1	/	10	Process	No	Absolute error, Relative error	131
Guo and Lyu (2000)	2000	1	Pascal, FORTRAN	11	Complexity	No	Type I, II error	41
Pedrycz et al. (2001a)	2001	10	Java, C++	8	Complexity	No	No indicator	20
Pedrycz et al. (2001b)	2001	1	Java	7	Complexity	No	No indicator	18
Zhong et al. (2004a)	2004	1	C++	13	Complexity	No	Error, FPR, FNR	123
Zhong et al. (2004b)	2004	2	C++	13	Complexity	No	Mean squared error, pure	9
Yang et al. (2006)	2006	2	Both C	12, 11	Complexity	No	Accuracy	8
Mahaweerawat et al. (2007)	2007	1	Not mentioned	11	Complexity	No	Accuracy, absolute residual	22
Yang et al. (2008)	2008	2	C, Pascal, FORTRAN	10, 7	Complexity	No	Accuracy, Type I, II error	16
Catal et al. (2009)	2009	3	C	29	Complexity	Yes	Error, FPR, FNR	93
Catal et al. (2010)	2010	3	C	29	Complexity	Yes	Error, FPR, FNR	19
Sandhu et al. (2010)	2010	1	Java	8	Complexity	No	Accuracy, FPR, FNR	10
Kaur et al. (2010)	2010	3	C++, C	8, 22	Complexity, Requirement	No	FPR, Recall	10
Kaur and Kumar (2011)	2011	1	Java	39	Complexity	No	Accuracy	4
Bishnu and Bhattacherjee (2012)	2012	3	C	29	Complexity	Yes	Error, FPR, FNR	160
Gupta et al. (2012b)	2012	3	C	4	Complexity	No	Meansquare error	2
Abaei et al. (2013)	2013	3	C	29	Complexity	Yes	Error, FPR, FNR	22
Gupta et al. (2013)	2013	2	C++	/	Complexity	No	Objective Function, Purity	5
Park and Hong (2014)	2014	3	C	29	Complexity	Yes	Accuracy, Error, FPR, FNR	18
Coelho et al. (2014)	2014	3	C++, C	21	Complexity	Yes	Accuracy	9
Pushpavathi et al. (2014)	2014	1	C	21	Complexity	No	Accuracy, RMSE, MAE, Reliability	1
Nam and Kim (2015)	2015	7	Java	465,26 (for 4,3 projects)	Network and change genealogy, Complexity	Yes	Precision Recall, F-measure, AUC	103
Yang and Qian (2016)	2016	16	Java	26,61,20 (for 3,5,8 projects)	Complexity, Process, previous-defect and entropy	Yes	Precision, Recall, and F-measure	7
Zhang et al. (2016)	2016	26	Java, C++, C	61,20 (for 5,21 projects)	The same as above	Yes	AUC	137
Yang et al. (2018)	2018	15	Java	26,61,20 (for 3,5,7 projects)	The same as above	Yes	Precision, Recall, and F-measure	1
Jothi (2018)	2018	5	C	29	Complexity	Yes	Error, FPR, FNR	1

a representative module or the average feature values of each cluster. If at least one element in the threshold vector is lower, the cluster is labeled as defective, otherwise as non-defective;

- Scheme 3 determines the label of each cluster based on some criteria, such as the risk level of the project, the defect number, the Bayesian rule, and module-order modeling;
- Scheme 4 denotes the IVS-based labeling strategy. After calculating the IVS values for all modules as stated in Section 2.1.8, the modules with the same IVS values are grouped into one cluster. This scheme ranks the clusters in descending order based on their IVS values, then labels the half top clusters as defective and others as non-defective. The process is described in the blue rectangle in Fig. 2;
- Scheme 5 denotes the defect-rate-based labeling strategy. This scheme ranks the modules based on their IVS values in descending order and calculates a threshold based on the defect rate, then labels the modules whose IVS values are greater than the threshold as defective and other modules as non-defective. The process is described in the purple rectangle in Fig. 2;
- Scheme 6 denotes the SFM based labeling strategy. This scheme clusters the modules into two groups and calculates the **S**um of **F**eature values of each **M**odule (**SFM**), then calculates the **A**verage value of the **SFM**s (**ASFM**) for all modules in each cluster. The cluster with larger ASFM is labeled as defective, and another cluster is labeled as non-defective. The process is depicted in Fig. 3.

The key differences between our work and the above studies are listed as follows: (1) we devoted to conduct a detailed analysis towards the clustering-based methods for UDP; (2) we used a larger-scale defect data as studied corpora; (3) our work was the first study to use several unexplored clustering-based methods (such as HCPC and HMC) to ensure that we select methods from a variety of families; (4) we were among the first to employ both traditional and effort-aware indicators to evaluate the performance of the CUDP methods; (5) we made the first step to analyze the interaction between the feature types and the performance of the CUDP methods.

3. Empirical study design

3.1. Comparative methods

To investigate if there exist any clustering based models that can outperform the supervised models for SDP, we chose some representative supervised models for comparison. Although one previous study (Ghotra et al., 2015) has investigated more than 30 supervised classification models for defect prediction, it is not suitable for us to consider all these models. As Hall et al. (2011) stated that simple classification models can also perform well on SDP task, in this work we just selected 6 off-the-shelf supervised models for comparison, including the probabilistic-based classifier **N**aive **B**ayes (**NB**), the statistic-based classifier **L**ogistic **R**egression (**LR**), the instance-based classifier *k*-**N**earest **N**eighbor (*k***NN**), the tree-based classifier **C**lassification **A**nd **R**egression **T**rees (**CART**), the rule-based classifier **R**epeated **I**ncremental

Table 4
A Summary of Previous Studies Related to CUDP.

Previous Studies	The used unsupervised models					Cluster number	LS
	PBC	DBC	MBC	GTBC	IVSBC		
Yuan et al. (2000)		Subtractive clustering				2	3
Guo and Lyu (2000)			EM			Determined by a criterion	3
Pedrycz et al. (2001a)			SOM			2	0
Pedrycz et al. (2001b)			SOM			2	0
Zhong et al. (2004a)	K-means		NG			20	1
Zhong et al. (2004b)	K-means		NG			20 or 30	2
Yang et al. (2006)	K-means, FCM		GMM			2 or 3	2
Mahaweerawat et al. (2007)			SOM			Determined by two parameters	1
Yang et al. (2008)				AP		2	1
Catal et al. (2009)	K-means					20	2
Catal et al. (2010)	X-means					Determined by optimizing	3
Kaur et al. (2010)	Two variants of K-means					2	0
Kaur and Kumar (2011)		DBSCAN				2	0
Sandhu et al. (2010)	K-means					2	0
Bishnu and Bhattacherjee (2012)	Quad-tree K-means					Heuristically determined	3
Gupta et al. (2012b)	FCM					Not mentioned	3
Abaei et al. (2013)			SOM			2	3
Gupta et al. (2013)	K-means, FCM					30, 15	0
Park and Hong (2014)	X-means		EM			Determined by optimizing	1
Coelho et al. (2014)	K-means		EM			2	0
Pushpavathi et al. (2014)	FCM and its variant					25	0
Nam and Kim (2015)					CLA, CLAMI	Based on IVS	4
Yang and Qian (2016)					ACL	2	5
Zhang et al. (2016)				SC		2	6
Yang et al. (2018)					CEL	2	5
Jothi (2018)	K-means, FCM,Quad-tree K-means					Not mentioned	0

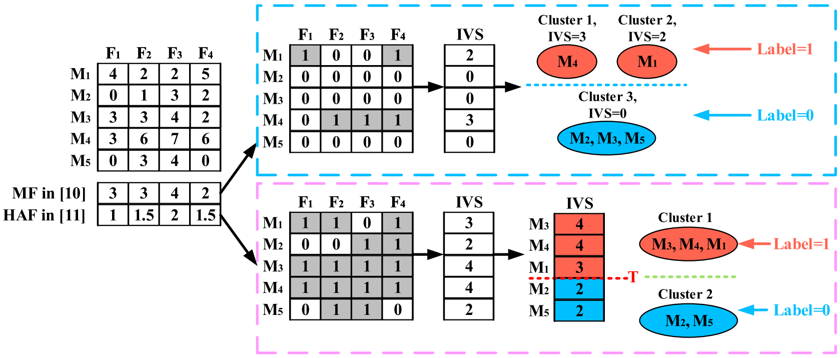


Fig. 2. The process of labeling scheme 4 and 5.

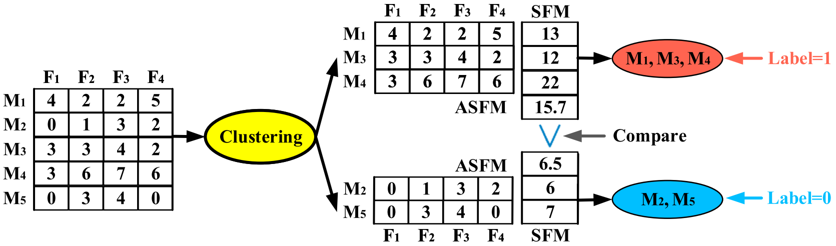


Fig. 3. The process of labeling scheme 6.

Pruning to **P**roduce **E**rror **R**eduction (**RIPPER**), and the ensemble-learning-based classifier **R**andom **F**orest (**RF**). The 6 models are typical and widely employed in previous SDP studies (Nam and Kim, 2015; Li et al., 2017; Xu et al., 2019c; Li et al., 2018) as the candidate of the basic classifiers and Zhang et al. (2016) compared their proposed unsupervised model with 4 out of the 6 supervised models. All these 6 models were implemented with the third-party functions in Weka library with the default parameters. The reasons are that: first, as the 40 unsupervised models in our empirical study were implemented using the default parameters without tuning the parameters, thus, it would be more appropriate to use the default values for the supervised model for a fair comparison; second, the main goal of this paper is to investigate the impacts of unsupervised models on the defect prediction performance, not to explore the influence of parameter tuning on the performance of supervised models, and previous studies have stated that parameter tuning is a time-consuming process in the field of software engineering (Arcuri and Fraser, 2013). Thus, in this work, we only reported the results of the supervised models with the default parameters.

3.2. Implementation for unsupervised methods

We implement 35 clustering methods with third-party functions in Weka, Python, and R libraries. Note that for the methods that are available in multiple libraries, we chose the implementation following the priority: Weka → Python → R. In addition, CLA and CLAMI in IVSBC family were implemented using the source code released by the authors, while ACL and CE methods in IVSBC family and SC method in GTBC family were reproduced by us following the corresponding descriptions in the original literatures.

3.3. Research **Q**uestion**s** (**RQs**)

In this work, we studied the following **R**esearch **Q**uestions (**RQ**).

RQ1: How do these selected methods perform on defect datasets with complexity features?

RQ2: How do these selected methods perform on defect datasets with process features?

RQ3: How do these selected methods perform on defect datasets with network features?

RQ4: How do these selected methods perform on defect datasets with all the aforementioned three types of features?

RQ5: What are the impacts of different feature types on the performance of the selected methods?

The first 3 questions explore the performance of clustering-based unsupervised models on defect data with individual feature types. The fourth question investigates the performance of these methods on defect data with combined features. The last question studies the impact of defect data with different feature types on the performance of these methods.

3.4. Benchmark dataset

As one goal of our empirical study is to investigate the impacts of feature types on CUDP performance, we chose a benchmark dataset released by Song et al. (2018). This benchmark dataset combines PROMISE dataset (Jureczko and Madeyski, 2010) and AEEEM dataset (D'Ambros et al., 2012) which have been widely used in previous defect prediction studies (Zhang et al., 2016; Zhou et al., 2018; Ghotra et al., 2015; Li et al., 2017, 2018; Jing et al., 2015, 2017). More specifically, this benchmark dataset includes 14 open-source software projects (9 projects from PROMISE dataset and 5 projects from AEEEM dataset) with a total of 27 versions in which 3 types of features are collected for each project version. Thus, we had a total of 81 project defect data. Table 5 presents the basic information of the defect data of these projects, including the link, the brief description, the version number, the total **S**um of the **L**ine **O**f **C**ode (**SLOC**), the total number of all modules (# Mod.), the number of defective modules (# Def.), and the percentage of defective modules (% Def.). The 3 types of features include 7 code complexity features, 11 process features, and 24 network features. Table 6 presents the brief definitions for these features. As all the projects were developed with Java language which may limit the generality of our work, more projects with other languages need to be included in our studied corpora.

3.5. Empirical study framework

Fig. 4 depicts the flow chart of our empirical study framework. For each feature type of one project version, we used the 1:1 stratified sampling technique to divide the data into part 1 and part 2. The stratified sampling strategy ensures that the defect ratios of the two parts are consistent with that of the original data. This division strategy has been used in previous defect prediction studies (Wang et al., 2016; Ryu et al., 2016; Xu et al., 2019c). In the first round, for supervised SDP, part 1 was fed into the 6 supervised models which were used to predict the labels of the modules in part 2. For CUDP, the 40 unsupervised models were only applied to part 2. In the second round, the two parts were swapped to run these methods again. This progress was repeated 50 times to alleviate the randomness bias of the data division. As a result, we obtained a total of 100 values for each indicator on each defect data and recorded the average values for performance analysis.

3.6. Labeling scheme

For the 40 unsupervised models, we followed the labeling scheme in Zhang et al. (2016) (i.e., Scheme 6 in Section 2.2) to label the clusters due to its simplicity and effectiveness. For the methods with predefined cluster number as 2, the labeling process is the same as that in Zhang et al. (2016), as depicted in Fig. 3. For the methods without predefined cluster numbers (i.e., multiple-cluster scenario), we used the labeling process in Fig. 5 to assign the labels to each cluster. More specifically, we first calculated the ASFMs for all clusters and the **M**ean values of these **ASFM**s (**MASFM**). Then, we labeled the clusters whose ASFMs are not less than MASFM as defective (i.e., the cluster including module M4), and label other clusters (i.e., the cluster including module M1 and M3, and the cluster including module M2 and M5) as non-defective. In other words, we used the average values on all features in each cluster to determine it class label. The motivation came from the heuristic rule of labeling two classes following the scheme in the previous work (Zhang et al., 2016) which suggested that the cluster with higher average feature values should be labeled as defective. This heuristic is based on the findings that larger or more complex files are mores like to contain defects than smaller files or the files with lower complexity (Nam and Kim, 2015; D'Ambros et al., 2012; Gaffney, 1984). Here, we gave an end to end example to explain the labeling process for the methods that group the modules into 2 clusters: for one data partition of ant-1.3 project with code complexity features, we first normalized the date in one part, then used the typical K-means method to group the normalized data into two clusters. The results show that one cluster contains 18 modules and one cluster contains 45 modules. The ASFM of the two clusters are 0.869 and −0.348, respectively. As the former one is larger than the latter one, we labeled all modules in the first cluster as defective and all modules in the second cluster as non-defective.

For the 6 supervised models, a classification threshold is needed for the learning methods to determine the labels of the modules. More specifically, a module is classified as defective if its probability given by the model is larger than the classification threshold, otherwise, it is classified as non-defective. In this work, we used the default threshold 0.5 as used in Zhou et al.'s work (Zhou et al., 2018).

Table 5
Description of the benchmark dataset.

Project	Description	Version	SLOC	# Mod.	# Def.	% Def.
Ant (http://ant.apache.org/)	A Java-based, shell independent build tool	1.3	37 699	125	20	16.00%
		1.4	54 195	178	40	22.47%
		1.5	87 047	293	32	10.92%
		1.6	113 246	351	92	26.21%
Camel (http://camel.apache.org/)	A integration framework based on Enterprise Integration Patterns	1.0	33 721	339	13	3.83%
		1.2	66 302	608	216	35.53%
		1.4	98 080	872	145	16.63%
		1.6	113 055	965	188	19.48%
ivy (http://ant.apache.org/ivy/)	A dependence manager focusing on flexibility and simplicity	2.0	87 769	352	40	11.36%
jedit (http://www.jedit.org/)	A cross platform programmer's text editor	3.2	128 883	272	90	33.09%
		4.0	144 803	306	75	24.51%
		4.1	153 087	312	79	25.32%
		4.2	170 683	367	48	13.08%
		4.3	202 363	492	11	2.24%
log4j (http://logging.apache.org/log4j/)	A logging package for printing log output	1.0	21 549	135	34	25.19%
poi (http://poi.apache.org/)	Java API for Microsoft documents format	2.0	93 171	314	37	11.78%
Synapse (http://synapse.apache.org/)	A lightweight and high-performance Enterprise Service Bus	1.0	28 806	157	16	10.19%
		1.1	42 302	222	60	27.03%
		1.2	53 500	256	86	33.59%
Velocity (http://velocity.apache.org/)	A template language engine	1.6	57 012	229	78	34.06%
xerces (http://xerces.apache.org/xerces-j/)	A Java-based XML parser	1.2	159 254	440	71	16.14%
		1.3	167 095	453	69	15.23%
Equinox framework (www.eclipse.org/equinox/)	An implementation of the OSGi core framework specification	3.4	39 534	324	129	39.81%
Eclipse JDT Core (www.eclipse.org/jdt/core/)	The Java infrastructure of the Java IDE	3.4	224 055	997	206	20.66%
Apache Lucene (https://lucene.apache.org)	A high-performance, full-featured text search engine library	2.4.0	73 184	691	64	9.26%
Mylyn (www.eclipse.org/mylyn/)	A task and application lifecycle management framework for Eclipse	3.1	156 102	1862	245	13.16%
Eclipse PDE UI (www.eclipse.org/pde/pde-ui/)	Providing a set of tools to create, develop, test, debug and deploy Eclipse plug-ins, fragments, features, update sites and RCP products	3.4.1	146 952	1497	209	13.96%

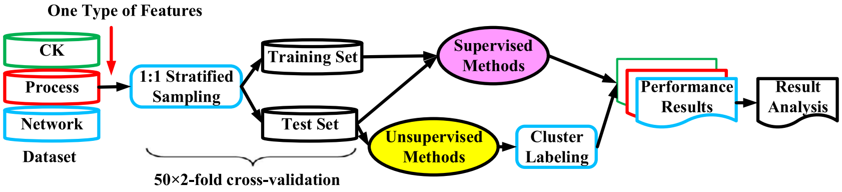


Fig. 4. Framework of our empirical study.

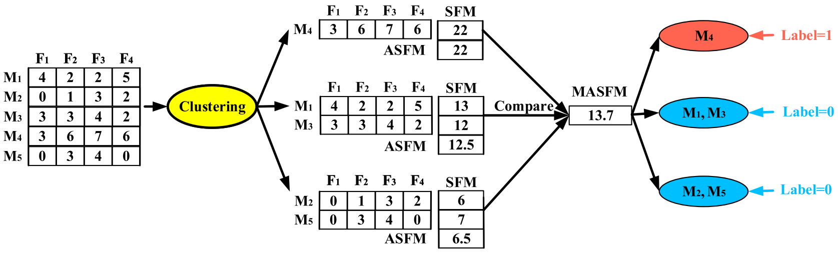


Fig. 5. Labeling scheme for multiple clusters.

3.7. Evaluation indicators

To measure the effectiveness of the total 46 methods for SDP, we employed 3 indicators as our performance measurement, including Matthew Correlation Coefficient (MCC), **EA**F-measure, and Popt. MCC is considered as the most appropriate indicator for SDP task (Song et al., 2018; Yao and Shepperd, 2020); EAF-measure is a more comprehensive effort-aware indicator recently proposed by Huang et al. (2017, 2018). Popt is a normalized version of the effort-aware indicator originally proposed in Mende and Koschke (2010).

We first defined 4 basic terms as follows:

Table 6
The brief definitions of the 3 types of features.

Feature type		Feature name	Brief description
Code complexity		Weighted methods per class (WMC)	The sum of the complexities of methods in a class
		Depth of Inheritance Tree (DIT)	The inheritance levels from the object hierarchy top for the class
		Number of Children (NOC)	The number of direct descendants of the class
		Coupling between object classes (CBO)	The number of classes coupled to a given class
		Response for a Class (RFC)	The number of different methods executed when an object receives a message
		Lack of cohesion in methods (LCOM)	The sets of methods not related through the sharing of some of the class's fields
		Lines of code (LOC)	The number of the lines of codes of the class
Process		Revisions	The number of revisions of a module
		Authors	The number of different authors that inspected a module
		Loc_added	Total number of lines of code added to a module for all revisions
		Max_loc_added	The maximum number of lines of code added to a module for all revisions
		Avg_loc_added	The average number of lines of code added to a module per revision
		Loc_deleted	Total number of lines of code deleted to a module for all revisions
		Max_loc_deleted	The maximum number of lines of code deleted to a module for all revisions
		Avg_loc_deleted	The average number of lines of code deleted to a module per revision
		Codechurn	Total number of lines of code changed to a module for all revisions
		Max_codechurn	The maximum number of lines of code changed to a module for all revisions
		Avg_codechurn	The average number of lines of code changed to a module per revision
Network	Ego	Size	The number of the nodes of the ego network
		Ties	The number of the edges involving in the network
		Pairs	The maximal number of directed ties
		Density	The percentage of the ties are actually presented
		WeakComp	The number of weak components in neighborhood
		nWeakComp	The normalized WeakComp by size
		TwoStepReach	The number of nodes within two directed steps of ego
		ReachEfficiency	The normalized TwoStepReach by Size
		Brokerage	The number of Pairs not directly connected
		nBrokerage	The normalized Brokerage by Pairs
		EgoBetweenness	The percentage of all geodesic paths among neighbors that pass through ego network
		nEgoBetweenness	The normalized EgoBetweenness by Size
	Structure	Effective Size (EffSize)	The number of alters connected to the ego minus the average degree of the alters
		Efficiency	The normalized EffSize by Size of the network
		Constraint	Measuring to what extend the ego is constraint by its alters
		Hierarchy	Measuring to what extent the constraint on ego is concentrated in a single alter
	Centrality	Degree	The number of nodes adjacent to a given node
		nDegree	The normalized Degree by the total number of nodes
		Closeness	The sum of the lengths of the shortest paths between a node and all other nodes
		Reachability	The number of nodes that a node can reach
		Eigenvector	Assigning relative scores to all nodes involving in the network
		nEigenvector	The normalized Eigenvector by the total number of nodes
		Betweenness	Measuring the frequency of a node appears on the shortest paths among other nodes
		nBetweenness	The normalized Betweenness by the total number of nodes

True **P**ositive (**TP**) and **F**alse **N**egative (**FN**) denote the number of defective modules that are correctly and incorrectly identified by a model, respectively; **T**rue **N**egative (**TN**) and **F**alse **P**ositive (**FP**) denote the number of non-defective modules that are correctly and incorrectly identified by a model, respectively.

(1) MCC. Given the above 4 terms, the general formula of MCC is defined as follows:

$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}} \tag{1}$$

MCC is non-effort-aware or traditional indicator since it does not consider the efforts of inspecting modules.

To evaluate the SDP performance in an effort-aware scenario (Mende and Koschke, 2010) in which only limited test resources are used for code review expecting the maximum profit (Yang et al., 2016; Arisholm et al., 2010; Kamei et al., 2013), we used 2 effort-aware indicators, i.e., EAF-measure and Popt. In previous studies, the number of LOC was used as proxy measure of the test resources involving in inspecting a module and the percentage of defective modules found after the inspection process was treated as the profit. In this work, we specified the test resources as 20% of total LOC following (Yang et al., 2016; Jiang et al., 2013; Yang et al., 2015; Xia et al., 2016). In the calculation process of EAF-measure, we employed the same ranking strategy in Xu et al. (2018), a variant version towards the strategy in Huang et al. (2018). The reason why we did not employ the ranking strategy in Huang et al. (2018) is that the probabilities of the modules being defective are not always available for unsupervised models. Fig. 6 depicts a diagram of the calculation process for the effort-aware indicators. The process consists of 5 main steps: (1) we clustered the modules into multiple groups (usually 2 groups) and labeled them as defective or non-defective based on the labeling strategy described in Section 3.6; (2) we ranked the modules in each cluster in ascending order based on their LOC values; (3) we concatenated the two ranked results in which the ranked result of the defective group is in the front of that of the non-defective group; (4) we simulated the developers or testers in inspecting the ranked modules until their cumulative LOC reached 20% (i.e., the cutoff point); and (5) we recorded statistics to calculate EAF-measure.

Before obtaining EAF-measure, we first needed to calculate **E**ffort-**A**ware Recall (**EA**Recall) and **E**ffore-**A**ware Precision (**EA**Precision). Given data with n_1 defective modules, after inspecting the ranked modules with 20% of LOC, we assumed n' modules and $n_1{}'$ actually defective modules have been detected. EARecall is defined as EARecall $= \mathrm{n}_{1'}/\mathrm{n}_1$ and EA-Precision is defined as $= \mathrm{n}_{1'}/\mathrm{n}'$.

(2) EAF-measure. Given the terms of EARecall and EAPrecision, the general formula of EAF-measure is defined as

$$\text{EAF-measure} = \frac{(1+\beta^2) \times \mathrm{EAPrecision} \times \mathrm{EARecall}}{\beta^2 \times \mathrm{EAPrecision} + \mathrm{EARecall}}. \tag{2}$$

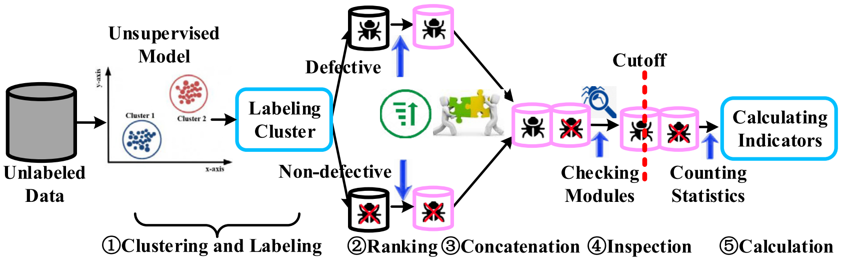


Fig. 6. Calculation process for effort aware indicators.

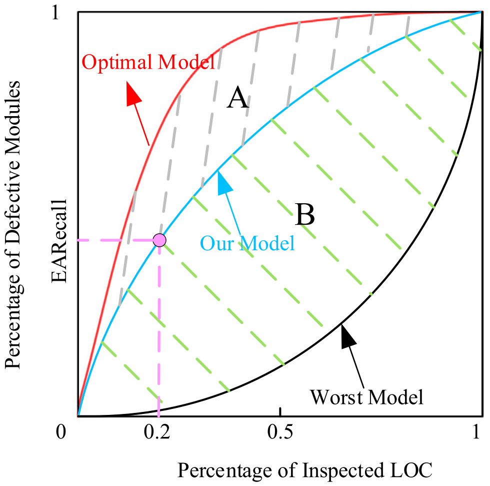


Fig. 7. LOC-based Alberg diagram.

In this work, we set β as 2 to emphasize more on the role of EARecall when balancing EARecall and EAPrecision following the previous studies (Xu et al., 2019b,a). In addition, we also upload the result of EAF-measure with β of 1 to our online supplementary materials.

(3) Popt. Another effort-aware indicator Popt is based on the area under the effort curve in an Alberg diagram (Arisholm et al., 2010). Fig. 7 presents an example of an LOC-based Alberg diagram. The calculation of Popt relies on 3 curves which correspond to an *optimal* model, our proposed model *m*, and a *worst* model. The 3 curves are described as follows:

- The *optimal* model means that all the modules are ranked in descending order, based on their actual defect density. In detail, the actual defective and non-defective model are ranked in ascending order according to their LOC respectively and the two ranked results were spliced, in which the ranked result of the defective group is in the front of that of the non-defective group.
- The proposed model *m* means that all modules are ranked according to our ranking strategy.
- The *worst* model means that all modules are ranked in ascending order, based on their actual defect density, that is, the results are opposite to that of the optimal model.

The Popt(m) is formally defined as follows:

$$\text{Popt}(m) = \frac{\text{Area}(m) - \text{Area}(\mathit{worst})}{\text{Area}(\mathit{optimal}) - \text{Area}(\mathit{worst})}. \tag{3}$$

where Area() represents the area under the corresponding curve.

According to this definition, Popt is equal to the ratio of the area of region B (the green dotted lines) to the sum of the area of region B and the region A (the gray dotted lines). A larger Popt value signifies that there is a smaller difference between our proposed model *m* and the *optimal* model.

3.8. Parameter configurations for unsupervised models

For the unsupervised models, if the clustering methods support specifying cluster number manually, we set it to 2, following the approach conducted by Zhang et al. (2016). Among the 40 selected unsupervised models, 4 of them i.e., MS, AP, SOM, and Cobweb, can determine the cluster number automatically. We hence did not specify the cluster number of them. For other parameters, we employ the default values in the Weka, Python, and R libraries.

3.9. Performance analysis method

In this work, we applied a statistical test technique, i.e., Friedman test with the improved Nemenyi post-hoc test in Herbold et al. (2018) (instead of the well-known novel Scott–Knott test) to analyze the performance results, which determines whether the performance differences among the methods are significant or simply due to the natural variability of the performance results (Hassan, 2009). The Friedman test is non-parametric which does not require the analysis data to follow a particular distribution and the improved Nemenyi test can divide the methods into non-overlapping groups. Whereas the novel Scott–Knott test (Ghotra et al., 2015; Xu et al., 2016a; Tantithamthavorn et al., 2017, 2018) requires the analysis data to satisfy the normality and homoscedasticity assumptions (Herbold, 2017), which is not always fulfilled in some cases. The combination of Friedman with Nemenyi test is widely adopted in previous SDP studies for significance test (Nam and Kim, 2015; Li et al., 2017, 2018; D'Ambros et al., 2012; Mende and Koschke, 2010; Herbold et al., 2018; Jiang et al., 2008; Lessmann et al., 2008).

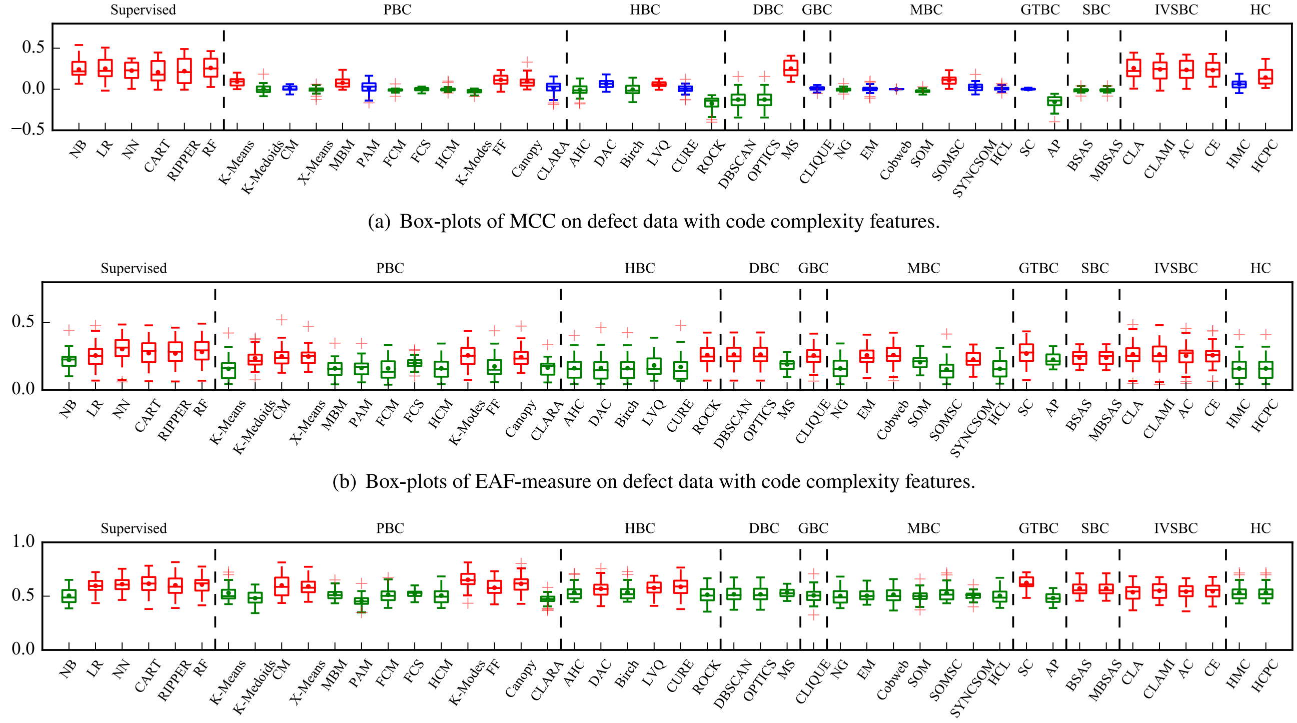
For the SDP study, if the *p* value of the Friedman test towards the performance results of multiple SDP methods is lower than 0.05, it denotes that these methods exist significant performance differences for SDP task. Then Nemenyi post-hoc test is employed to distinguish which SDP methods are significantly different from others.

4. Empirical results

4.1. Results for RQ1

Since we needed to perform a total of 46 methods (40 unsupervised models and 6 supervised models) on 27 defect data with 100 times, we obtained 124200 (46 × 27 × 100) records of the performance results for this question.

Fig. 8 depicts the box-plots of 3 indicators on defect data with code complexity features. We reported both the average and median indicator values represented by the colored point and bands inside the boxes, respectively. The boxes with different



(a) Box-plots of MCC on defect data with code complexity features.

(b) Box-plots of EAF-measure on defect data with code complexity features.

(c) Box-plots of Popt on defect data with code complexity features.

Fig. 8. Box-plots of the 3 indicator values across 27 defect data with code complexity features.

colors imply distinct meanings as follows: the red boxes indicate that the corresponding methods belong to the top-ranked group after conducting the statistical test. In other words, these methods outperform the others with a statistical significance; the green boxes indicate that the corresponding methods belong to the bottom-ranked group, which implies that these methods are outperformed by others with a statistical significance; the blue boxes indicate that the corresponding methods belong to the middle-ranked group.

From Fig. 8, we can observe that, first, in terms of the supervised model family, 5 classifiers except for NB belong to the top-ranked group on all indicators. In terms of the PBC family, one method (i.e., Canopy) belong to the top-ranked group on all indicators, and 4 methods (i.e., CM, X-means, K-Modes, and Canopy) belong to the top-ranked group on 2 effort-aware indicators. In terms of the HBC, DBC, GBC, MBC, and HC families, no methods belong to the top-ranked group on at least 2 indicators. In terms of the GTBC family, one method (i.e., SC) belongs to the top group on 2 effort-aware indicators. In terms of the SBC, all two methods (i.e., BSAS and MBSAS) belong to the top-ranked group on 2 effort-aware indicators. In terms of the IVSBC family, all methods belong to the top-ranked group on all indicators.

In terms of MCC, all classifiers in the supervised model family, 4 methods in the PBC family, one method in HBC, DBC, MBC, and HC families, and all methods in the IVSBC family belong to the top-ranked group. In terms of EAF-measure, 5 methods in supervised model and PBC families, one method in the HBC family, 2 methods in the DBC family, one method in the GBC family, 3 methods in the MBC family, one method in the GTBC family, and all methods in SBC and IVSBC families belong to the top-ranked group. In terms of Popt, 5 methods in supervised model and PBC families, 3 method in the HBC family, one method in the GTBC family, and all methods in SBC and IVSBC families belong to the top-ranked group.

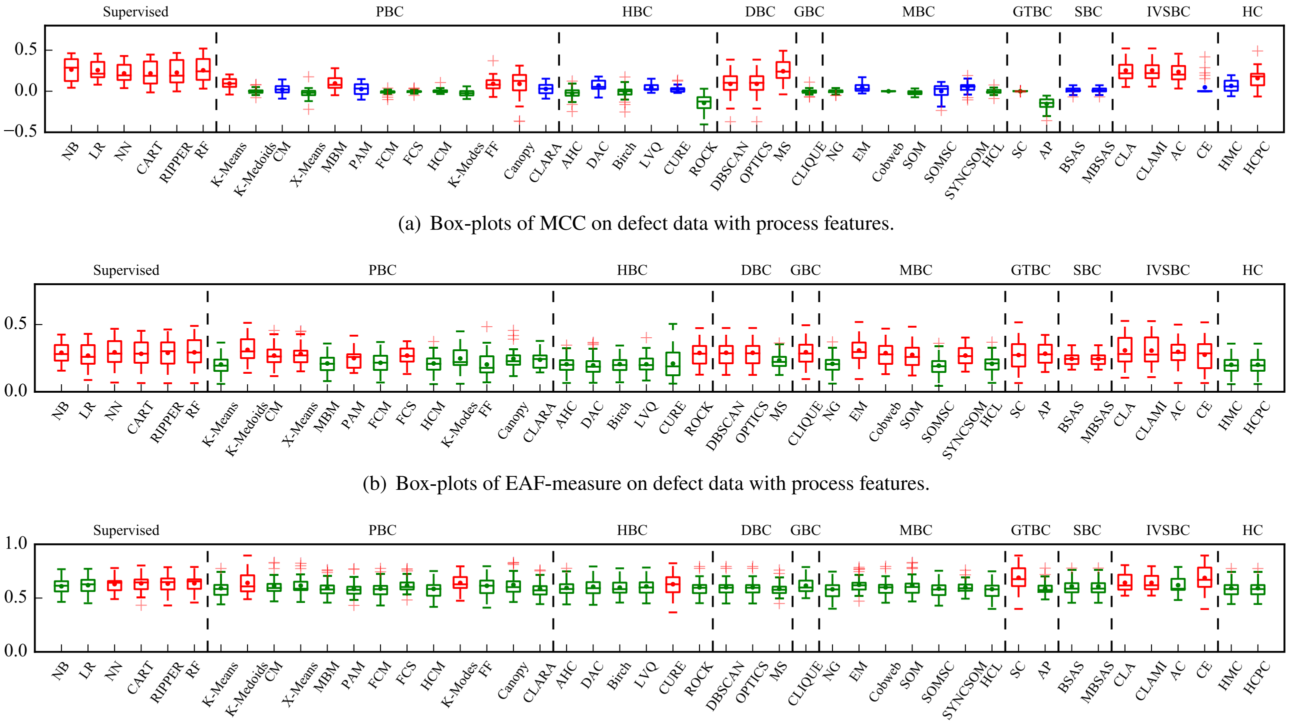
To sum up, on defect data with code complexity features, 5 classifiers except for NB in the supervised model family, Canopy in PBC family, and all methods in IVSBC family perform best on all indicators.

4.2. Results for RQ2

Since we also needed to perform a total of 46 methods on 27 defect data with 100 times, we obtain 124,200 (46 × 27 × 100) records of the performance results for this question.

Fig. 9 depicts the box-plots of 3 indicators on defect data with process features. From Fig. 9, we observe that: first, in terms of the supervised model family, 4 classifiers except for NB and LR belong to the top-ranked group on all indicators, NB and LR classifiers belong to the top-ranked group on one traditional and one effort-aware indicators. In terms of the PBC family, no methods belong to the top-ranked group on all indicators, one method (i.e., K-Medoids) belongs to the top-ranked group on 2 effort-aware indicators. In terms of HBC, GBC, MBC, SBC, and HC families, no methods belong to the top-ranked group on at least 2 indicators. In terms of the DBC family, 2 methods (i.e, DBSCAN and OPTICS) belong to the top-ranked group on one traditional and one effort-aware indicators. In terms of the GTBC family, one method (i.e., SC) belongs to the top-ranked group on 2 effort-aware indicators. In terms of the IVSBC family, two methods (i.e., CLA and CLAMI) belong to the top-ranked group on all indicators, one method (i.e., CE) belongs to the top-ranked group on 2 effort-aware indicators, and one method (i.e., AC) belongs to the top-ranked group on one traditional and one effort-aware indicators.

In terms of MCC, all classifiers in the supervised model family, 4 methods in the PBC family, all methods in the DBC family, 3 methods in the IVSBC family, and one method in the HC family belong to the top-ranked group. In terms of EAF-measure, all classifiers in the supervised model family, 5 methods in the PBC family, one method in the HBC family, 2 methods in the DBC family, one method in the GBC family, 4 methods in the MBC family, all methods in GTBC, SBC, and IVSBC families belong to the



(a) Box-plots of MCC on defect data with process features.

(b) Box-plots of EAF-measure on defect data with process features.

(c) Box-plots of Popt on defect data with process features.

Fig. 9. Box-plots of 3 indicator values across 27 defect data with process features.

top-ranked group. In terms of Popt, 4 classifiers in the supervised model family, 2 methods in the PBC family, one method in HBC and GTBC families, 3 methods in the IVSBC family belong to the top-ranked group.

Overall, on defect data with process features, 4 classifiers (except for NB and LR) in the supervised model family, CLA and CLAMI in the IVSBC family achieve the best performance on all indicators.

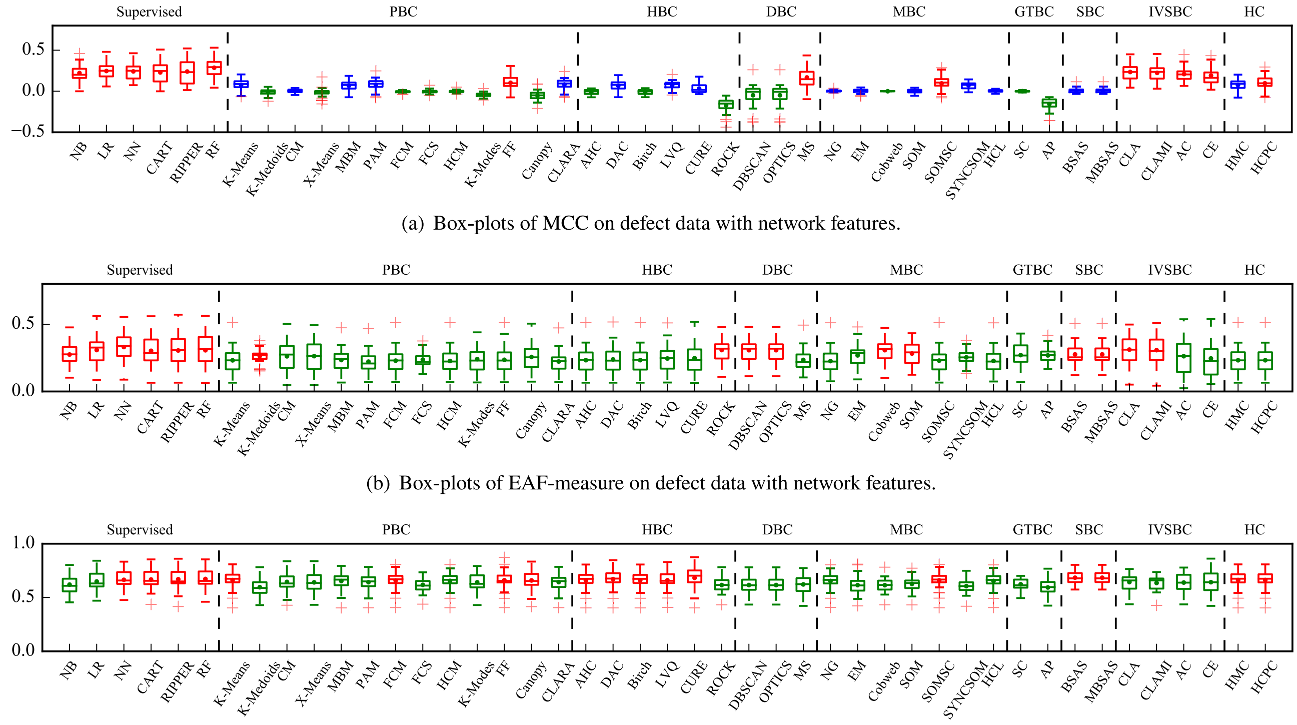
4.3. Results for RQ3

Due to the practical difficulties in launching the CLIQUE method in the GBC family with network features, we have to ignore CLIQUE from this study. The process of CLIQUE is that: it first divides each dimension into a certain number of equal-width grid cells and saves those whose density is greater than a threshold as clusters; then each set of two dimensions is examined: if there are two intersecting cells in these 2 dimensions and the density in the intersection is greater than the threshold, the intersection is also saved as a cluster. This is repeated for all sets (e.g., 3 dimensions, 4 dimensions) until the total feature dimension (Hassani, 2015). From this point of view, CLIQUE is faced with the curse of dimensionality, which means that the complete enumeration of all subspaces becomes intractable with the increasing dimensionality. Our experiments show that we could not apply the CLIQUE method to our defect data with 24 network features due to the required run time. For example, on project Eclipse JDT Core, CLIQUE needs nearly 3000 s (50 min) for one run of one data split. Since there are in total 100 runs, CLIQUE needs 5000 min (nearly 3.5 days). As we have 27 projects, it can be roughly estimated that we need nearly 3 months to get the results for CLIQUE on defect data with network metrics. Considering the practical applicability of CLIQUE, we did not consider CLIQUE in this question since infinite time is not always available for the SDP task. As a result, we performed in total 45 methods (39 unsupervised models and 6 supervised models) on 27 defect data with 100 times, and obtained 121,500 (45 × 27 × 100) records of the performance results for this question.

Fig. 10 depicts the box-plots of 3 indicators on defect data with network features. From Fig. 10, we have the following findings: first, in terms of the supervised model family, 4 classifiers except for NB and LR belong to the top-ranked group on all indicators, NB and LR classifiers belong to the top-ranked group on one traditional and one effort-aware indicators. In terms of PBC, HBC, and DBC families, no methods belong to the top-ranked group on at least 2 indicators. In terms of the MBC family, one method (i.e., SOMSC) belong to the top-ranked group on one traditional and one effort-aware indicators. In terms of the GTBC family, all methods belong to the bottom-ranked group on all indicators. In terms of the SBC family, all methods belong to the top-ranked group on 2 effort-aware indicators. In terms of the IVSBC family, two methods (i.e., CLA and CLAMI) belong to the top-ranked group on one traditional and one effort-aware indicators. In terms of the HC family, one method (i.e., HCPC) belongs to the top-ranked group on one traditional and one effort-aware indicators.

In terms of MCC, all methods in the supervised model family, one method in PBC, DBC, MBC, and HC families, all methods in the IVSBC family belong to the top-ranked group. In terms of EAF-measure, all classifiers in the supervised model family, one method in PBC and HBC families, 2 methods in DBC and MBC families, all methods in the SBC family, and 2 methods in the IVSBC family belong to the top-ranked group. In terms of Popt, 4 classifiers in the supervised model family, 4 methods in the PBC family, 5 methods in the HBC family, one method in the MBC family, and all methods in SBC and HC families belong to the top-ranked group.

In summary, on defect data with network features, 4 classifiers (except for NB and LR) in the supervised model family exhibits the best superiority on all indicators.

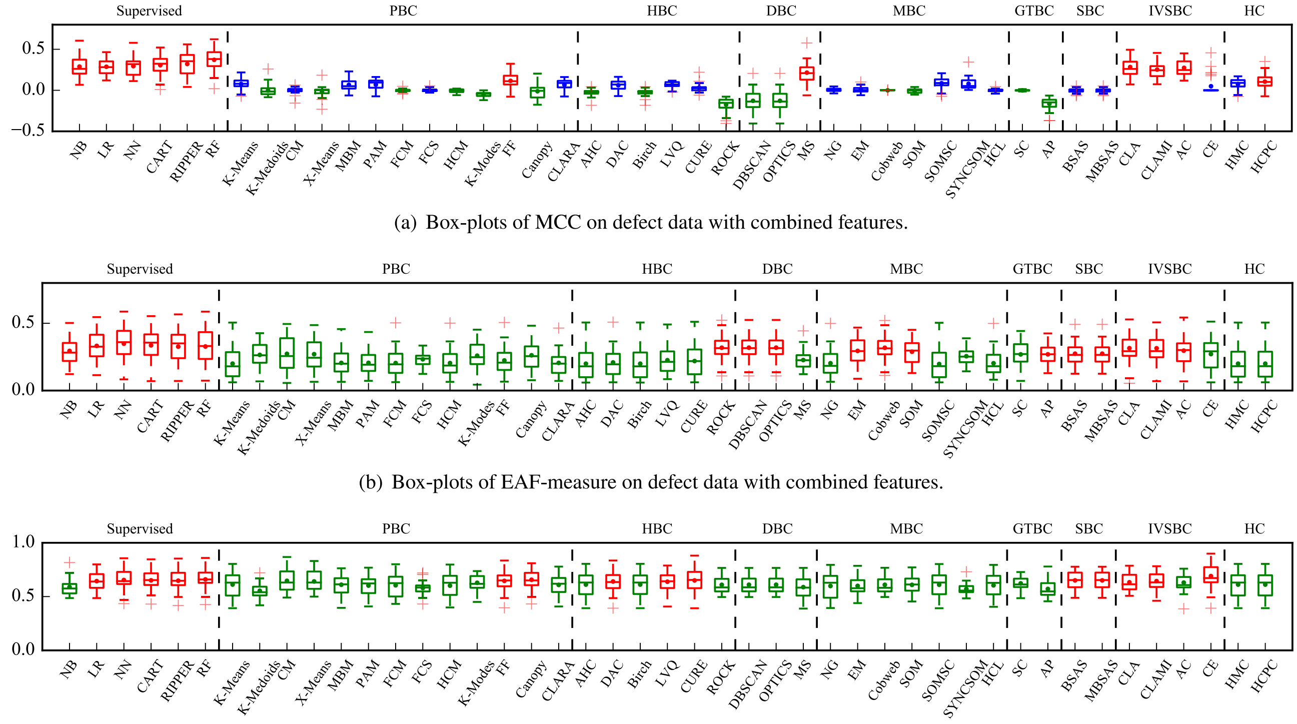


(a) Box-plots of MCC on defect data with network features.

(b) Box-plots of EAF-measure on defect data with network features.

(c) Box-plots of Popt on defect data with network features.

Fig. 10. Box-plots of 3 indicator values across 27 defect data with network features.



(a) Box-plots of MCC on defect data with combined features.

(b) Box-plots of EAF-measure on defect data with combined features.

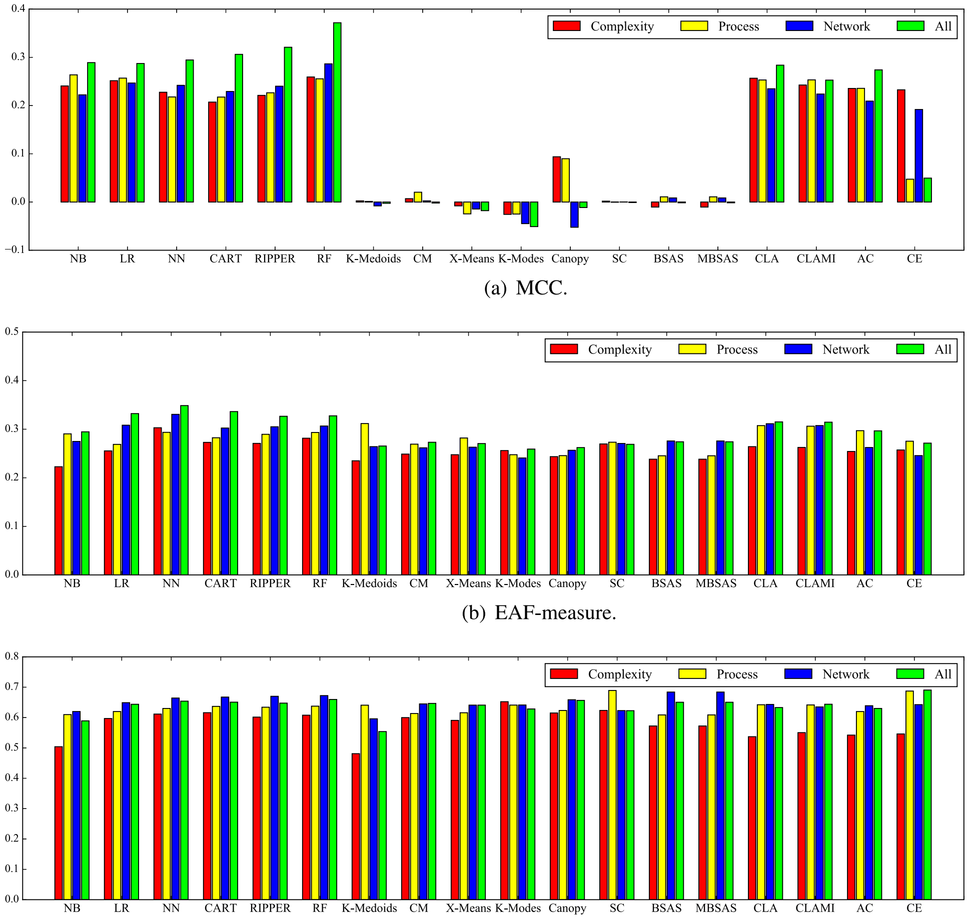
(c) Box-plots of Popt on defect data with combined features.

Fig. 11. Box-plots of 3 indicator values across 27 defect data by combining the 3 types of features.

4.4. Results for RQ4

Since the dimension of the combined features is larger than that of the network features, we also could not get the performance of the CLIQUE method on the defect data with combined features. Thus, we also did not consider CLIQUE in this question. Again, we performed in total 45 methods on 27 defect data with 100 times, and obtained 121,500 (45 × 27 × 100) records of the performance results for this question.

Fig. 11 depicts box-plots of 3 indicators on defect data with all features by combining code complexity, process, and network features. From Fig. 11, we have the following findings: first, in



(a) MCC.

(b) EAF-measure.

(c) Popt.

Fig. 12. Average values of 3 indicator for the selected methods on defect data with different feature types and the combined features.

terms of the supervised model family, 5 classifiers except for NB belong to the top-ranked group on all indicators. In terms of the PBC family, one method (i.e., FF) belongs to the top-ranked group on one traditional and one effort-aware indicators. In terms of HBC, DBC, MBC, GTBC and HC families, no methods belong to the top-ranked group on at least 2 indicators. In terms of the SBC family, all 2 methods belong to the top-ranked group on 2 effort-aware indicators. In terms of the IVSBC family, 2 methods (i.e., CLA and CLAM) belong to the top-ranked group on all indicators, one method (i.e., AC) belongs to the top-ranked group on one traditional and one effort-aware indicators.

In terms of MCC, all classifiers in the supervised model family, one method in PBC, DBC, HC families, and 3 methods in the IVSBC family belong to the top-ranked group. In terms of EAF-measure, all classifiers in the supervised model family, one method in the HBC family, 2 methods in the DBC family, 3 methods in the MBC family, one method in the GTBC family, all methods in the SBC family, and 3 methods in the IVSBC family belong to the top-ranked group. In terms of Popt, 5 classifiers in the supervised model family, 2 methods in the PBC family, 3 methods in the HBC family, all methods in the SBC family, and 3 methods in the IVSBC family belong to the top-ranked group.

Overall, on defect data with all features, 5 classifiers (except for NB) in the supervised model family, CLA and CLAMI in the IVSBC family perform significantly better on all indicators.

4.5. Results for RQ5

To answer this question, we considered the unsupervised models who belong to the top-ranked group on all indicators or on 2 effort-aware indicators over defect data with one of the feature types and all supervised models. According to the result analysis in the above 4 research questions, 12 unsupervised models were remained (i.e., K-Medoids, CM, X-Means, K-Modes, and Canopy in PBC family, SC in GTBC family, and all methods in SBC and IVSBC families). Thus, we used 18 models (12 unsupervised + 6 supervised models) to analyze this question.

Fig. 12 shows bar charts of the average values of 3 indicators for the selected methods on defect data with different feature types and the combined features. From this figure, we can observe that, in terms of the 6 classifiers in supervised model family, they achieve the best average performance on Popt over defect data with network features and the best average performance on other 2 indicators over defect data with the combined features. For the methods except for the ones in the IVSBC family, they perform bad on the traditional indicator but perform well on 2 effort-aware indicators over defect data with different types of features. In terms of the 12 unsupervised models, their performance of different indicators vary according to the feature types of the defect data. For example, for Canopy, it achieves better performance on the traditional indicator over defect data with code complexity and process features, but obtains better performance on 2 effort-aware indicators over defect data with network and combined features; for the two methods in the SBC family (i.e., BSAS and MBSAS), they achieves the best MCC values on defect data with process and network features, the best EAF-measure values over defect data with network and combined features, the best Popt values over defect data with network features; for SC, it achieves nearly the same MCC and EAF-measure values on defect data with different feature types, but obtains the best Popt value over defect data with process features. For CLA, CLAMI, and AC, they perform the best on the traditional indicator over defect data with combined features, and do not perform well on 2 effort-aware indicators over defect data with code complexity features. In addition, CLA and CLAMI obtain similar average performance on 2 effort-aware indicators over defect data with process, network, and combined features. For CE, it does not perform well on the traditional indicator but performs the best on 2 effort-aware indicators over defect data with process and combined features, and it obtains nearly the same average performance on all indicators over defect data with process and combined features.

From the above observations, the superiority of the selected 18 methods (especially for the unsupervised models) on defect data with distinct feature types varies according to the indicators used.

5. Discussion

5.1. Implications

We provided some implications from the analysis of our experimental results for practitioners and researchers.

(1) **The methods in the HBC, GBC, and HC families should be avoided in practice** for defect prediction. The reason is that no methods from the above families perform well on all indicators and on 2 effort-aware indicators over defect data with any kind of feature types and the combined features. This may explain why the methods in these 3 families were not explored in previous studies. We recommended that practitioners should avoid using such methods when conducting SDP on unlabeled defect data.

(2) **The methods in IVSBC family appear to be optimal options for CUDP**. Overall, they present promising performance in most cases. As these methods design specific rules (such as the violation score) which rely on the defect data characteristics to divide the modules, they are able to well adapt to the SDP task in practical applications.

(3) **Clustering-based defect prediction models should be highly regarded for researchers**. Our experimental results show that several clustering-based models are not inferior to the classical supervised models, such as Canopy method in the PBC family which can achieve competitive performance or even better performance over defect data with code complexity features. As unsupervised models do not require the prior knowledge of the defect data by label collection which is known to be time-consuming and labor-intensive (Fu and Menzies, 2017; Xu et al., 2019d; Chen et al., 2015), they can promote the quality assurance activity.

(4) **Selection of clustering-based models for CUDP should comprehensively consider feature types of the defect data and the used indicators**. Performance of these methods varies towards the two factors. For example, the effort-aware performance of Canopy prefers to the defect data with network and combined features while the traditional performance of Canopy prefers to the defect data with other two types of features. We recommend that software engineering researchers should extract suitable features from the source code for specific performance according to actual requirements.

(5) **A combination of features does not always enable the defect data to promote the performance of unsupervised models**. Although the supervised models achieve better performance on two indicators (i.e., MCC and EAF-measure) over defect data with combined features overall, the clustering-based unsupervised models do not always perform well on such defect data. For example, defect data with combined features are not suitable to K-Medoids on 2 effort-aware indicators, and to Canopy on the traditional indicator. Thus, when the researchers hesitate whether to combine different feature types to form a new defect dataset, the decision should rely on the used unsupervised methods and indicators.

(6) As there exist some clustering-based models with promising defect prediction performance, **we can use them with the labeling scheme to annotate some data for the researchers and practitioners to perform some other supervised learning tasks**, expecting to save the cost of manual annotation.

(7) **As discussed in** Section 3.2**, the implementation of our framework (with the help of Weka, Python and R) is quite generic**. Hence, apart from comparing clustering-based defect prediction models, we believe that our framework, with slight modifications, could be also applied to other comprehensive comparative studies concerning clustering-based approaches.

5.2. Threats to validity

In this subsection, we presented the following 3 major threats to the validity of our work.

(1) **External Validity**: Our experiments were conducted using publicly available benchmark data from 27 versions of 14 open source projects. An external validity threat is that all of these projects were developed with Java language and we do not consider the projects developed with other languages, such as C, C++ or python. This may limit the generality of our experimental results. In addition, since our benchmark data consists of 3 types of features, i.e., code complexity features, process features, and network features, our experimental results may not be generalized to the defect data with other feature types, such as text features (Scandariato et al., 2014) and developer's scattering features (Di Nucci et al., 2018). Future experiments on various defect data can alleviate such threats.

(2) **Internal Validity**: For method implementation, we used the third-party library implementation or the code provided by the authors for most methods to avoid potential mistakes in the implementation process by ourselves, which is beneficial to relieve the threat to the internal validity. One potential threat is that our implementations for AC and CE may be slightly inconsistent with the original versions. In this work, two graduate students participated in checking the source code to minimize this threat. For the parameter setting of the cluster number, we set it to 2 for most methods. Thus, another threat is that the derived results may exist a certain degree of differences for other settings of this parameter. More fine-tuned parameters would be needed in future studies.

(3) **Construct Validity**: The threat to the construct validity is that the used performance indicators may not provide a comprehensive evaluation for the methods. In this work, we used one traditional and 2 effort-aware indicators to measure the performance of these methods. Although these indicators were commonly used in the defect prediction domain, we still cannot claim that our conclusions are consistent with that of other indicators that we have not analyzed in this study. Another threat is the appropriateness of the used statistical test technique. In this work, we used a non-parametric test, the Friedman test with

Nemenyi post-hoc test to check the significant differences among these methods. This test is a classic statistical test which is employed in many previous defect prediction studies. Rather than using the original test, we used the improved version proposed in Herbold et al. (2018) which is more suitable to generate non-overlapping groups for statistical analysis.

6. Related work

The topic of SDP has been an active research field and been widely studied in the last two decades. Recent studies on this topic can be roughly divided into 3 categories. The first category is that the researchers employed ready-made techniques or proposed new methods for SDP task in which machine learning methods are the mainstream trends. This type of studies aims to improve the performance of detecting the defective modules, such as the work in Jing et al. (2014), Xia et al. (2016) and Jing et al. (2015). The second category is that researchers collected different features from the source code for SDP tasks. This type of studies aims to extract more effective representations for the modules to promote the identification of the defective modules, such as the work in Moser et al. (2008), Jiang et al. (2013), Jureczko and Spinellis (2010), Di Nucci et al. (2017) and Yan et al. (2020). The third category concerns the works leveraging previous publications or dataset to perform comprehensive comparisons on the performance of a set of methods. The first two categories mainly focus on improving the SDP performance from a technological perspective while the last one mainly focuses on conducting literature reviews or empirical studies to investigate the impacts of different experimental components on the prediction performance, such as the work in Ghotra et al. (2015), Song et al. (2018) and Xu et al. (2016a). Our work belongs to the last category. In this section, we report the research progress about this category.

6.1. Empirical studies on classification models for SDP

Lessmann et al. (2008) considered three potential factors that may cause bias for SDP performance, including the used classification models, the used performance indicators, and the statistical tests used for empirical findings. To investigate this issue, they choose 22 classifiers as studied objects and applied them to 10 publicly available projects from original NASA dataset. Besides, they employed AUC to evaluate the performance of these classifiers and used the Friedman test with the Nemenyi test to analyze the results. The results showed that there exist no significant performance differences among the top 17 classifiers. Following Lessmann et al.'s work, Ghotra et al. (2015) conducted a larger-scale empirical study for a total of 31 classification models on 29 projects from 3 datasets (i.e., the original NASA dataset, the cleaned NASA dataset, and the PROMISE dataset). They employed the AUC and a double Scott–Knott test to evaluate and analyze the performance of these classifiers, respectively. They found that the results are similar to those in Lessmann et al. (2008) on original NASA dataset, but the results on the other 2 datasets show a statistically distinct separation among these classifiers. They hence concluded that the choice of classification models have impacts on SDP performance. Tantithamthavorn et al. (2018) explored the impacts of parameter optimization on 26 classification models on 4 datasets with 12 performance indicators. They found that the optimization can improve the AUC performance of models by up to 40 percents.

Different from the above studies which focused on analyzing the impacts of supervised classification models on the SDP performance, in this work, we investigated the impacts of clustering-based unsupervised models on the SDP performance.

6.2. Empirical studies on unsupervised models for SDP

Yang et al. (2016) were the first to compare the performance of unsupervised models with that of supervised models for JIT defect prediction. Their results on 6 projects showed that some simple unsupervised models achieved better effort-aware performance than supervised models under 3 prediction scenarios. Fu and Menzies (2017) revisited Yang et al.'s work and proposed a supervised model, called OneWay which is based on the implication of Yang et al.'s simple unsupervised model. They repeat experiment on the same project as Yang et al.'s work and the results showed that OneWay performed better than Yang et al.'s unsupervised models. They suggested that simple supervised models should be given priority to perform defect prediction task. Yan et al. (2017) replicated Yang et al.'s work for file-level defect prediction task. The results showed that their conclusion was consistent with Yang et al.'s under the cross-project prediction scenario but was contrary to Yang et al.'s under the within-project prediction. Huang et al. (2017, 2018) also replicated Yang et al.'s work and analyzed the reason why the unsupervised model could achieve better effort-aware performance. They proposed a simple supervised model called CBS (Huang et al., 2017) and CBS+ (Huang et al., 2018) that performed better than the unsupervised model in terms of two effort-aware indicators and could inspect fewer changes. Chen et al. (2019) made a first attempt to compare the performance between unsupervised models and supervised models for predicting the defect number. They conducted experiments on 7 projects with 24 versions under 3 prediction scenarios and suggested that the unsupervised method should be treated as the baseline method when researchers proposed new supervised defect number prediction models.

Different from the above studies in which the unsupervised models ranked the software modules according to the feature values, in this work, we focused on the unsupervised models based on the clustering techniques that group the software modules into different clusters. Recently, Li et al. (2020) conducted a systematic review of unsupervised models for SDP. They mainly analyzed some experimental results in existing articles. Different from their work, we did a more detailed summary for the experimental configurations of existing articles, like the information of used datasets, performance indicators, and labeling schemes. In addition, we conducted large-scale experiments to comprehensively analyze the SDP performance of 40 clustering-based model and investigated the impacts of feature types on the SDP performance.

6.3. Empirical studies on feature selection and reduction methods for SDP

Muthukumaran et al. (2015) investigated 7 ranking-based, 2 wrapper-based and one embedded-based feature selection methods on the original NASA dataset and AEEEM dataset. They found that the performance of the 10 methods has no significant differences. Gao et al. (2011) studied 7 ranking-based feature selection methods followed by 4 feature subset searching strategies on a private dataset. They found that 6 ranking-based methods obtained similar performance. Wang et al. (2011) conducted an empirical study on 6 ranking-based and 2 ensemble-based feature selection methods on 3 datasets. They found that the performance of the ranking-based methods is affected by 2 factors (i.e., the datasets and classification models used) while the ensemble-based methods are stable and robust to the 2 factors.

Xu et al. (2016a) empirically studied 32 feature selection methods from 5 families on 3 datasets (i.e., the original NASA dataset, the cleaned NASA dataset, and the AEEEM dataset) with a random forest classifier. They used the same performance indicators and statistic test as in Ghotra et al. (2015). They found that these methods have significant performance differences on each dataset. Following Xu et al.'s work, Ghotra et al. (2017) conducted a larger-scale empirical comparison for 30 feature selection methods on 2 datasets (i.e., the clean NASA dataset and the AEEEM dataset) with 21 classification models. They found that the correlation-based filter subset selection method with the BestFirst search strategy performed the best, and the performance impacts of these methods vary across the used classification models and the datasets. Kondo et al. (2019) performed an empirical study to investigate the impact of 8 feature reduction techniques on the performance of 5 classification models and 5 clustering models over 3 datasets. The difference between feature selection and feature reduction methods is that the former one reduces the number of features by choosing a subset based on the importance degrees of the features, while the latter one reduces the number of features by generating new or combining features through feature transformation methods. They found that neural network-based feature reducing methods (i.e., restricted Boltzmann machine and auto-encoder) performed the best on clustering models, and created features with small variants in performance across the classification models and clustering models.

The above studies explored the application of feature selection or reduction methods for SDP task, which usually need the labeled defect data to select the informative feature subset. Different from these studies, our work concentrated on the usage of clustering-based unsupervised models in SDP without involving in the feature engineering techniques.

6.4. Empirical studies on sampling-based imbalanced learning technologies for SDP

There are different methods to alleviate the class imbalance issue for SDP, such as the sampling-based, ensemble-based, and cost-sensitive-based imbalanced learning methods. The sampling-based methods add or remove some modules to rebalance the training set. Ensemble-based methods combine the decisions of multiple classifiers to obtain better performance than the single one. Cost-sensitive-based methods take the misclassification costs for different classes into consideration by treating different misclassification differently. That is, the cost for labeling a defective module as non-defective is higher than the cost for labeling a non-defective module as defective. Many empirical studies about the imbalanced SDP issue focus on sampling-based methods.

Kamei et al. (2007) examined the impacts of 4 sampling methods on the SDP performance of 4 classification models over 2 industry legacy software systems. They found that these sampling methods are only helpful to improve the performance of linear and logistic models. Bennin et al. (2016) explored the impacts of 4 sampling methods on the effort-aware SDP performance of 10 classification models over 10 software projects from PROMISE dataset. They found that these sampling methods could promote the performance of all the models when the defect percentage of the data is between 20% and 30%. Bennin et al. (2017b) investigated the impacts of 6 sampling methods on the SDP performance of 5 classification models over 10 software projects from PROMISE dataset. They found that these methods had statistic and practical significances in terms of false positives, Recall, G-mean, but not in terms of AUC. Bennin et al. (2017a) studied the impacts of a configurable parameter (i.e., the defect percentage of the data) on the SDP performance of 7 sampling methods with 5 classification models over 10 projects from PROMISE dataset. They found that this parameter indeed affects the performance of these models in terms of the used indicators except for AUC.

Tantithamthavorn et al. (2018) assessed the impacts of 4 sampling methods on the performance and interpretation of 7 classification models with 10 performance indicators over 101 projects from 5 datasets. They found that the optimized SMOTE method and under-sampling method could increase the performance of Recall and AUC, but are not helpful to interpret the models. Song et al. (2018) systematically evaluated 17 imbalanced learning methods (including sampling-based, ensemble-based, cost-sensitive-based and imbalanced ensemble-based methods) with 7 classification models over 27 software projects. They found that these methods are more effective on the defect data with moderate or higher imbalance rates, and a particular combination of the imbalance learning methods and classification models is important for improving the performance of SDP.

The sampling-based imbalanced learning methods need the label information to balance the training sets for learning unbiased supervised classification models. Different from the above studies, our work assumed that the label information was not available and studied the unsupervised models that do not consider the class imbalance processing.

6.5. Literature reviews on SDP studies

Some researches have surveyed a large amount of SDP studies and designed some criteria to identify the primary ones. They mainly analyzed these articles to find common patterns and give some deep insights.

Catal and Diri (2009) reviewed 74 articles about SDP and found that the usage of publicly available datasets, the method-level features and machine learning methods are the mainstream trends. Hall et al. (2011) performed an in-depth analysis of the quantitative and qualitative results of 36 articles with sufficient contextual and methodological information selected from 208 articles. Their empirical observations suggested that simple classification model and the combination of process, product, and people-based features tended to perform well and the feature selection methods were beneficial to the SDP performance. Shepperd et al. (2014) conducted a meta-analysis on 600 sets of prediction results published in 42 primary studies. They found that the choice of classification models had little impacts on the SDP performance. In contrast, the researcher group was the major explanatory factor to affect the SDP performance.

Hosseini et al. (2017) conducted a systematic literature review towards **C**ross **P**roject **D**efect **P**rediction (**CPDP**) articles and identified 30 primary studies. CPDP utilizes the labeled data of external projects to build a classifier to predict the labels of the unlabeled data in the project at hand. They pointed out the most commonly-used performance indicators, the well-performed classification models and the widely-used datasets, and suggested that more attention should be paid on CPDP as it is still a challenging task. In order to identify which CPDP method performed the best, Herbold et al. (2018) replicated 24 existing CPDP methods and evaluated them on 5 datasets. They found that 3 methods achieved the best performance in most cases and pointed out that there is still room for improvement before the CPDP methods can be put into practice. Similarly, Porto et al. (2018) implemented 31 state-of-the-art CPDP methods and compared them on 47 versions of 15 projects from PROMISE dataset. They identified 4 methods that achieved the best performance across datasets and proposed a meta-learning solution to dynamically choose the suitable method for a specific project.

Different from the above studies which mainly paid attention to review the SDP work under the supervised scenarios, i.e., the defect prediction task within the project or across projects, in this work, we conducted literature reviews only for the defect prediction studies under the unsupervised scenario.

7. Conclusions

We conducted a large-scale comparison to analyze SDP performance differences among 40 clustering-based unsupervised models and 6 typical supervised models. We made the first step towards investigating the impacts of the feature types of defect data on the performance of these methods. Our experimental results on 81 defect data indicate that not all clustering-based unsupervised models are worse than the supervised models, and the performance of the methods in the IVSBC family is particularly outstanding overall. Moreover, we observed that the feature types can indeed affect the performance of the studied methods on different indicators.

As of our future work, we plan to explore the impacts of feature selection on the SDP performance of these clustering-based models, since previous studies have empirically demonstrated that different feature selection methods can significantly affect the SDP performance of supervised models (Xu et al., 2016a; Ghotra et al., 2017). In addition, as previous studies stated that the class imbalance issue of the defect data has negative impacts on the SDP performance of supervised models (Tantithamthavorn et al., 2018; Song et al., 2018), we will also explore how to consider this issue in the clustering-based models to further improve their performance.

We provide the benchmark dataset, the experimental scripts, and experimental results at https://github.com/sailer2020/CUDP.

CRediT authorship contribution statement

Zhou Xu: Writing - original draft, Methodology, Data curation. **Li Li:** Conceptualization, Writing - review & editing. **Meng Yan:** Supervision, Formal analysis. **Jin Liu:** Supervision. **Xiapu Luo:** Writing - review & editing. **John Grundy:** Writing - review & editing. **Yifeng Zhang:** Methodology, Software, Visualization. **Xiaohong Zhang:** Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors would like to thank Jaechang Nam for providing the source code of CLA and CLAMI. This work is supported by the National Key Research and Development Project (No. 2018YFB2101200), the National Natural Science Foundation of China (Nos. 62002034, 61972290), the Fundamental Research Funds for the Central Universities (Nos. 2020CDJQY-A021, 2020CD-CGRJ072), China Postdoctoral Science Foundation (No. 2020M673 137), the Natural Science Foundation of Chongqing in China (No. cstc2020jcyj-bshX0114), the Hong Kong Research Grant Council Project (No. 152239/18E). Grundy is supported by Australian Research Council Laureate Fellowship FL190100035.

References

Abaei, G., Rezaei, Z., Selamat, A., 2013. Fault prediction by utilizing self-organizing map and threshold. In: Proceedings of the 7th International Conference on Control System, Computing and Engineering. IEEE, pp. 465–470.

Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P., 1998. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications, Vol. 27. ACM.

Alboukadel, K., 2017a. Hierarchical k-means clustering. http://www.sthda.com/english/articles/30-advanced-clustering/100-hierarchical-k-means-clustering-optimize-clusters/.

Alboukadel, K., 2017b. Hierarchical clustering on principal components. http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/117-hcpc-hierarchical-clustering-on-principal-components-essentials/#algorithm-of-the-hcpc-method.

Ankerst, M., Breunig, M.M., Kriegel, H.-P., Sander, J., 1999. Optics: ordering points to identify the clustering structure. In: ACM Sigmod Record, Vol. 28. ACM, pp. 49–60.

Arcuri, A., Fraser, G., 2013. Parameter tuning or default values? an empirical investigation in search-based software engineering. Empir. Softw. Eng. 18 (3), 594–623.

Arisholm, E., Briand, L.C., Johannessen, E.B., 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. J. Syst. Softw. 83 (1), 2–17.

Béjar Alonso, J., 2013. K-Means Vs Mini Batch K-Means: A Comparison. Tech. Rep.

Bennin, K.E., Keung, J., Monden, A., 2017a. Impact of the distribution parameter of data sampling approaches on software defect prediction models. In: Proceedings of the 24th Asia-Pacific Software Engineering Conference. APSEC, IEEE, pp. 630–635.

Bennin, K.E., Keung, J., Monden, A., Kamei, Y., Ubayashi, N., 2016. Investigating the effects of balanced training and testing datasets on effort-aware fault prediction models. In: Proceedings of the 40th Annual Computer Software and Applications Conference, Vol. 1. COMPSAC, IEEE, pp. 154–163.

Bennin, K.E., Keung, J., Monden, A., Phannachitta, P., Mensah, S., 2017b. The significant effects of data sampling approaches on software defect prioritization and classification. In: Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM, IEEE Press, pp. 364–373.

Bezdek, J.C., Ehrlich, R., Full, W., 1984. Fcm: The fuzzy c-means clustering algorithm. Comput. Geosci. 10 (2–3), 191–203.

Bishnu, P.S., Bhattacherjee, V., 2012. Software fault prediction using quad tree-based k-means clustering algorithm. IEEE Trans. Knowl. Data Eng. (TKDE) 24 (6), 1146–1150.

Catal, C., Diri, B., 2009. A systematic review of software fault prediction studies. Expert Syst. Appl. 36 (4), 7346–7354.

Catal, C., Sevim, U., Diri, B., 2009. Clustering and metrics thresholds based software fault prediction of unlabeled program modules. In: Proceedings of the 6th International Conference on Information Technology. IEEE, pp. 199–204.

Catal, C., Sevim, U., Diri, B., 2010. Metrics-Driven Software Quality Prediction without Prior Fault Data. In: Lecture Notes in Electrical Engineering, vol. 60, pp. 189–199.

Chen, L., Fang, B., Shang, Z., Tang, Y., 2015. Negative samples reduction in cross-company software defects prediction. Inf. Softw. Technol. 62, 67–77.

Chen, X., Zhang, D., Zhao, Y., Cui, Z., Ni, C., 2019. Software defect number prediction: Unsupervised vs supervised methods. Inf. Softw. Technol. 106, 161–181.

Cheng, Y., 1995. Mean shift, mode seeking, and clustering. IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 17 (8), 790–799.

Coelho, R.A., dos RN Guimarães, F., Esmin, A.A., 2014. Applying swarm ensemble clustering technique for fault prediction using software metrics. In: Proceedings of the 13th International Conference on Machine Learning and Applications. ICMLA, IEEE, pp. 356–361.

D'Ambros, M., Lanza, M., Robbes, R., 2012. Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empir. Softw. Eng. 17 (4–5), 531–577.

Dave, R.N., 1990. Fuzzy shell-clustering and applications to circle detection in digital images. Int. J. Gen. Syst. 16 (4), 343–355.

Dempster, A.P., Laird, N.M., Rubin, D.B., 1977. Maximum likelihood from incomplete data via the em algorithm. J. R. Stat. Soc. 1–38.

Di Nucci, D., Palomba, F., De Rosa, G., Bavota, G., Oliveto, R., De Lucia, A., 2017. A developer centered bug prediction model. IEEE Trans. Softw. Eng. (TSE) 44 (1), 5–24.

Di Nucci, D., Palomba, F., De Rosa, G., Bavota, G., Oliveto, R., De Lucia, A., 2018. A developer centered bug prediction model. Trans. Softw. Eng. 44 (1), 5–24.

Ding, C., He, X., 2002. Cluster merging and splitting in hierarchical clustering algorithms. In: Proceedings of the 2nd International Conference on Data Mining. ICDM, IEEE, pp. 139–146.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al., 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Vol. 96. KDD. pp. 226–231.

Fisher, D.H., 1987. Knowledge acquisition via incremental conceptual clustering. Mach. Learn. 2 (2), 139–172.

Frey, B.J., Dueck, D., 2007. Clustering by passing messages between data points. Science 315 (5814), 972–976.

Fritzke, B., 1997. Some Competitive Learning Methods. Artificial Intelligence Institute, Dresden University of Technology.

Fu, W., Menzies, T., 2017. Revisiting unsupervised learning for defect prediction. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, FSE. pp. 72–83.

Gaffney, J.E., 1984. Estimating the number of faults in code. IEEE Trans. Softw. Eng. (TSE) (4), 459–464.

Gao, K., Khoshgoftaar, T.M., Wang, H., Seliya, N., 2011. Choosing software metrics for defect prediction: an investigation on feature selection techniques. Softw. - Pract. Exp. 41 (5), 579–606.

Geremia, S., Tamburri, D.A., 2018. Varying defect prediction approaches during project evolution: A preliminary investigation. In: Proceedings of the 2nd IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation. IEEE, pp. 1–6.

Ghotra, B., McIntosh, S., Hassan, A.E., 2015. Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proceedings of the 37th International Conference on Software Engineering. ICSE, IEEE, pp. 789–800.

Ghotra, B., McIntosh, S., Hassan, A.E., 2017. A large-scale study of the impact of feature selection techniques on defect classification models. In: Proceedings of the 14th International Conference on Mining Software Repositories. MSR, IEEE, pp. 146–157.

Guha, S., Rastogi, R., Shim, K., 1998. Cure: an efficient clustering algorithm for large databases. In: ACM Sigmod Record, Vol. 27. ACM, pp. 73–84.

Guha, S., Rastogi, R., Shim, K., 2000. Rock: A robust clustering algorithm for categorical attributes. Inf. Syst. 25 (5), 345–366.

Guo, P., Lyu, M.R., 2000. Software quality prediction using mixture models with em algorithm. In: Proceedings of the 1st Asia-Pacific Conference on Quality Software. IEEE, pp. 69–78.

Gupta, D., Goyal, V.K., Mittal, H., 2012a. Analysis of clustering techniques for software quality prediction. In: Proceedings of the 2nd International Conference on Advanced Computing & Communication Technologies. IEEE, pp. 6–9.

Gupta, D., Goyal, V., Mittal, H., 2012b. Software quality analysis of unlabeled program moduls with fuzzy-c means clustering techniques. IMRS's Int. J. Eng. Sci. 1 (2), Published.

Gupta, D., Goyal, V.K., Mittal, H., 2013. Estimating of software quality with clustering techniques. In: 2013 Third International Conference on Advanced Computing and Communication Technologies. IEEE, pp. 20–27.

Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S., 2011. A systematic literature review on fault prediction performance in software engineering. IEEE Trans. Softw. Eng. (TSE) 38 (6), 1276–1304.

Han, J., Pei, J., Kamber, M., 2011. Data Mining: Concepts and Techniques. Elsevier.

Hartigan, J.A., Wong, M.A., 1979. Algorithm as 136: A k-means clustering algorithm. J. R. Stat. Soc. 28 (1), 100–108.

Hassan, A.E., 2009. Predicting faults using the complexity of code changes. In: Proceedings of the 31st International Conference on Software Engineering. ICSE, IEEE Computer Society, pp. 78–88.

Hassani, M., 2015. Package 'subspace'. https://cran.r-project.org/web/packages/subspace/subspace.pdf.

Herbold, S., 2017. Comments on scottknottesd in response to an empirical comparison of model validation techniques for defect prediction models. IEEE Trans. Softw. Eng. (TSE) 43 (11), 1091–1094.

Herbold, S., Trautsch, A., Grabowski, J., 2018. A comparative study to benchmark cross-project defect prediction approaches. IEEE Trans. Softw. Eng. (TSE) 44 (9), 811–833.

Hochbaum, D.S., Shmoys, D.B., 1985. A best possible heuristic for the k-center problem. Math. Oper. Res. 10 (2), 180–184.

Hosseini, S., Turhan, B., Gunarathna, D., 2017. A systematic literature review and meta-analysis on cross project defect prediction. IEEE Trans. Softw. Eng. (TSE) 45 (2), 111–147.

Huang, Z., 1997. A fast clustering algorithm to cluster very large categorical data sets in data mining. In: Workshop on Research Issues on Data Mining and Knowledge Discovery, Vol. 3, No. 8. pp. 34–39.

Huang, Q., Xia, X., Lo, D., 2017. Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction. In: Proceedings of the 33rd International Conference on Software Maintenance and Evolution. ICSME, IEEE, pp. 159–170.

Huang, Q., Xia, X., Lo, D., 2018. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. Empir. Softw. Eng. 1–40.

Jiang, Y., Cukic, B., Ma, Y., 2008. Techniques for evaluating fault prediction models. Empir. Softw. Eng. 13 (5), 561–595.

Jiang, T., Tan, L., Kim, S., 2013. Personalized defect prediction. In: Proceedings of the 28th International Conference on Automated Software Engineering. ASE, IEEE, pp. 279–289.

Jin, X., Han, J., 2016. K-medoids clustering. In: Encyclopedia of Machine Learning and Data Mining. Springer, pp. 1–3.

Jing, X., Wu, F., Dong, X., Qi, F., Xu, B., 2015. Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning. In: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. FSE, ACM, pp. 496–507.

Jing, X.-Y., Wu, F., Dong, X., Xu, B., 2017. An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems. IEEE Trans. Softw. Eng. (TSE) 43 (4), 321–339.

Jing, X.-Y., Ying, S., Zhang, Z.-W., Wu, S.-S., Liu, J., 2014. Dictionary learning based software defect prediction. In: Proceedings of the 36th International Conference on Software Engineering. ICSE, ACM, pp. 414–423.

Jothi, R., 2018. A comparative study of unsupervised learning algorithms for software fault prediction. In: 2018 Second International Conference on Intelligent Computing and Control Systems. ICICCS, IEEE, pp. 741–745.

Jureczko, M., Madeyski, L., 2010. Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering. ACM, p. 9.

Jureczko, M., Spinellis, D., 2010. Using object-oriented design metrics to predict software defects. In: Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej, pp. 69–81.

Karegowda, A.G., Jayaram, M., Manjunath, A., 2012. Cascading k-means clustering and k-nearest neighbor classifier for categorization of diabetic patients. Int. J. Eng. Adv. Technol. 1 (3), 147–151.

Kainulainen, J., Kainulainen, J.J., 2002. Clustering Algorithms: Basics and Visualization. Helsinki University of Technology, Laboratory of Computer and Information Science.

Kamei, Y., Monden, A., Matsumoto, S., Kakimoto, T., Matsumoto, K.-i., 2007. The effects of over and under sampling on fault-prone module detection. In: Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement. ESEM, IEEE, pp. 196–204.

Kamei, Y., Shihab, E., Adams, B., Hassan, A.E., Mockus, A., Sinha, A., Ubayashi, N., 2013. A large-scale empirical study of just-in-time quality assurance. IEEE Trans. Softw. Eng. (TSE) 39 (6), 757–773.

Kaufman, L., Rousseeuw, P.J., 2009. Finding Groups in Data: An Introduction to Cluster Analysis, Vol. 344. John Wiley & Sons.

Kaur, D., Kaur, A., Gulati, S., Aggarwal, M., 2010. A clustering algorithm for software fault prediction. In: Proceedings of th International Conference on Computer and Communication Technology. IEEE, pp. 603–607.

Kaur, A., Kaur, K., Kaur, H., 2015. An investigation of the accuracy of code and process metrics for defect prediction of mobile applications. In: Proceedings of the 4th International Conference on Reliability, Infocom Technologies and Optimization. IEEE, pp. 1–6.

Kaur, S., Kumar, D., 2011. Quality prediction of object oriented software using density based clustering approach. Int. J. Eng. Technol. 3 (4), 440.

Kohonen, T., 1995. Learning vector quantization. In: Self-Organizing Maps. Springer, pp. 175–189.

Kohonen, T., 1998. The self-organizing map. Neurocomputing 21 (1–3), 1–6.

Kondo, M., Bezemer, C.-P., Kamei, Y., Hassan, A.E., Mizuno, O., 2019. The impact of feature reduction techniques on defect prediction models. Empir. Softw. Eng. 1–39.

Lessmann, S., Baesens, B., Mues, C., Pietsch, S., 2008. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Trans. Softw. Eng. (TSE) 34 (4), 485–496.

Li, Z., Jing, X.-Y., Wu, F., Zhu, X., Xu, B., Ying, S., 2018. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. Autom. Softw. Eng. 25 (2), 201–245.

Li, Z., Jing, X.-Y., Zhu, X., Zhang, H., Xu, B., Ying, S., 2017. On the multiple sources and privacy preservation issues for heterogeneous defect prediction. IEEE Trans. Softw. Eng. (TSE).

Li, N., Shepperd, M., Guo, Y., 2020. A systematic review of unsupervised learning techniques for software defect prediction. Inf. Softw. Technol. 106287.

MacQueen, J., et al., 1967. Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1. Oakland, CA, USA. pp. 281–297.

Mahaweerawat, A., Sophatsathit, P., Lursinsap, C., 2007. Adaptive self-organizing map clustering for software fault prediction. In: Proceedings of the 4th International Joint Conference on Computer Science and Software Engineering. pp. 35–41.

Martinetz, T., Schulten, K., et al., 1991. A neural-gas network learns topologies. Artif. Neural Netw. 397–402.

McCallum, A., Nigam, K., Ungar, L.H., 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In: Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining. KDD, ACM, pp. 169–178.

Mende, T., Koschke, R., 2010. Effort-aware defect prediction models. In: Proceedings of the 14th European Conference on Software Maintenance and Reengineering. CSMR, IEEE, pp. 107–116.

Moser, R., Pedrycz, W., Succi, G., 2008. 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. ICSE, ACM, pp. 181–190.

Muthukumaran, K., Rallapalli, A., Murthy, N., 2015. Impact of feature selection techniques on bug prediction models. In: Proceedings of the 8th India Software Engineering Conference. ACM, pp. 120–129.

Nam, J., Kim, S., 2015. Clami: Defect prediction on unlabeled datasets. In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 452–463.

Ng, A.Y., Jordan, M.I., Weiss, Y., 2002. On spectral clustering: Analysis and an algorithm. In: Advances in Neural Information Processing Systems. NIPS, pp. 849–856.

Novikov, A., 2018. annoviko/pyclustering: pyclustering 0.8.2 release. Nov.

Novikov, A., Benderskaya, E.N., 2014. Sync-som: double-layer oscillatory network for cluster analysis. In: Proceedings of the 3rd International Conference on Pattern Recognition Applications and Methods. pp. 305–309.

Park, M., Hong, E., 2014. Software fault prediction model using clustering algorithms determining the number of clusters automatically. Int. J. Softw. Eng. Appl. 8.

Pedrycz, W., Succi, G., Musílek, P., Bai, X., 2001a. Using self-organizing maps to analyze object-oriented software measures. J. Syst. Softw. 59 (1), 65–82.

Pedrycz, W., Succi, G., Reformat, M., Musilek, P., Bai, X., 2001b. Self organizing maps as a tool for software analysis. In: Proceedings of the 14th Canadian Conference on Electrical and Computer Engineering, Vol. 1. IEEE, pp. 93–97.

Pelleg, D., Moore, A.W., et al., 2000. X-means: Extending k-means with efficient estimation of the number of clusters. In: Proceedings of the 17th International Conference on Machine Learning, Vol. 1. ICML. pp. 727–734.

Porto, F., Minku, L., Mendes, E., Simao, A., 2018. A systematic study of cross-project defect prediction with meta-learning. arXiv preprint arXiv:1802.06025.

Pushpavathi, T., Suma, V., Ramaswamy, V., 2014. Analysis of software fault and defect prediction by fuzzy c-means clustering and adaptive neuro fuzzy c-means clustering. Int. J. Sci. Eng. Res. 5 (9).

Radjenović, D., Heričko, M., Torkar, R., Živkovič, A., 2013. Software fault prediction metrics: A systematic literature review. Inf. Softw. Technol. 55 (8), 1397–1418.

Ryu, D., Choi, O., Baik, J., 2016. Value-cognitive boosting with a support vector machine for cross-project defect prediction. Empir. Softw. Eng. 21 (1), 43–71.

Sandhu, P.S., Singh, J., Gupta, V., Kaur, M., Manhas, S., Sidhu, R., 2010. A K-Means Based Clustering Approach for Finding Faulty Modules in Open Source Software Systems, Vol. 72. World Academy of Science, Engineering and Technology, pp. 654–658.

Scandariato, R., Walden, J., Hovsepyan, A., Joosen, W., 2014. Predicting vulnerable software components via text mining. IEEE Trans. Softw. Eng. (TSE) 40 (10), 993–1006.

Shepperd, M., Bowes, D., Hall, T., 2014. Researcher bias: The use of machine learning in software defect prediction. IEEE Trans. Softw. Eng. (TSE) 40 (6), 603–616.

Song, Q., Guo, Y., Shepperd, M., 2018. A comprehensive investigation of the role of imbalanced learning for software defect prediction. IEEE Trans. Softw. Eng. (TSE) 45 (12), 1253–1269.

Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2017. An empirical comparison of model validation techniques for defect prediction models. IEEE Trans. Softw. Eng. (TSE) 43 (1), 1–18.

Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2018. The impact of automated parameter optimization on defect prediction models. IEEE Trans. Softw. Eng. (TSE).

Theodoridis, S., Koutroumbas, K., 2006. Pattern Recognition, third ed. Academic Press, London.

Wang, H., Khoshgoftaar, T.M., Van Hulse, J., Gao, K., 2011. Metric selection for software defect prediction. Int. J. Softw. Eng. Knowl. Eng. 21 (02), 237–257.

Wang, T., Zhang, Z., Jing, X., Zhang, L., 2016. Multiple kernel ensemble learning for software defect prediction. Autom. Softw. Eng. 23 (4), 569–590.

Xia, X., Lo, D., Pan, S.J., Nagappan, N., Wang, X., 2016. Hydra: Massively compositional model for cross-project defect prediction. IEEE Trans. Softw. Eng. (TSE) 42 (10), 977–998.

Xu, Z., Li, S., Luo, X., Liu, J., Zhang, T., Tang, Y., Xu, J., Yuan, P., Keung, J., 2019a. Tstss: A two-stage training subset selection framework for cross version defect prediction. J. Syst. Softw. 154, 59–78.

Xu, Z., Li, S., Xu, J., Liu, J., Luo, X., Zhang, Y., Zhang, T., Keung, J., Tang, Y., 2019b. Ldfr: Learning deep feature representation for software defect prediction. J. Syst. Softw. 158, 110402.

Xu, Z., Liu, J., Luo, X., Yang, Z., Zhang, Y., Yuan, P., Tang, Y., Zhang, T., 2019c. Software defect prediction based on kernel pca and weighted extreme learning machine. Inf. Softw. Technol. 106, 182–200.

Xu, Z., Liu, J., Luo, X., Zhang, T., 2018. Cross-version defect prediction via hybrid active learning with kernel principal component analysis. In: Proceedings of the 25th International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 209–220.

Xu, Z., Liu, J., Yang, Z., An, G., Jia, X., 2016a. The impact of feature selection on defect prediction performance: An empirical comparison. In: Proceedings of the 27th International Symposium on Software Reliability Engineering. ISSRE, IEEE, pp. 309–320.

Xu, Z., Pang, S., Zhang, T., Luo, X.-P., Liu, J., Tang, Y.-T., Yu, X., Xue, L., 2019d. Cross project defect prediction via balanced distribution adaptation based transfer learning. J. Comput. Sci. Technol. 34 (5), 1039–1062.

Xu, Z., Xuan, J., Liu, J., Cui, X., 2016b. Michac: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering. In: Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering, Vol. 1. SANER, IEEE, pp. 370–381.

Yan, M., Fang, Y., Lo, D., Xia, X., Zhang, X., 2017. File-level defect prediction: Unsupervised vs. supervised models. In: Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM, IEEE, pp. 344–353.

Yan, M., Xia, X., Fan, Y., Hassan, A.E., Lo, D., Li, S., 2020. Just-in-time defect identification and localization: A two-phase framework. IEEE Trans. Softw. Eng. (TSE).

Yang, X., Lo, D., Xia, X., Zhang, Y., Sun, J., 2015. Deep learning for just-in-time defect prediction. In: Proceedings of the International Conference on Software Quality, Reliability and Security. QRS, IEEE, pp. 17–26.

Yang, J., Qian, H., 2016. Defect prediction on unlabeled datasets by using unsupervised clustering. In: Proceedings of 18th International Conference on 18th IEEE International Conference on High Performance Computing and Communications; 14th International Conference on Smart City; 2nd International Conference on Data Science and Systems. IEEE, pp. 465–472.

Yang, Y., Yang, J., Qian, H., 2018. Defect prediction by using cluster ensembles. In: Proceedings of the 10th International Conference on Advanced Computational Intelligence. IEEE, pp. 631–636.

Yang, B., Yin, Q., Xu, S., Guo, P., 2008. Software quality prediction using affinity propagation algorithm. In: Proceedings of 2008 International Joint Conference on Neural Networks. IJCNN, IEEE, pp. 1891–1896.

Yang, B., Zheng, X., Guo, P., 2006. Software metrics data clustering for quality prediction. In: International Conference on Intelligent Computing. Springer, pp. 959–964.

Yang, Y., Zhou, Y., Liu, J., Zhao, Y., Lu, H., Xu, L., Xu, B., Leung, H., 2016. Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. FSE, ACM, pp. 157–168.

Yao, J., Shepperd, M., 2020. Assessing software defection prediction performance: Why using the matthews correlation coefficient matters. In: Proceedings of the Evaluation and Assessment in Software Engineering, EASE. pp. 120–129.

Yuan, X., Khoshgoftaar, T.M., Allen, E.B., Ganesan, K., 2000. An application of fuzzy clustering to software quality prediction. In: Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology. IEEE, pp. 85–90.

Zhang, T., Ramakrishnan, R., Livny, M., 1996. Birch: an efficient data clustering method for very large databases. In: ACM Sigmod Record, Vol. 25. ACM, pp. 103–114.

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

Zhong, S., Khoshgoftaar, T.M., Seliya, N., 2004a. Unsupervised learning for expert-based software quality estimation. In: Proceedings of 8th International Symposium on High-Assurance Systems Engineering. HASE, Citeseer, pp. 149–155.

Zhong, S., Khoshgoftaar, T.M., Seliya, N., 2004b. Expert-based software measurement data analysis with clustering techniques. In: IEEE Intelligent Systems, Special Issue on Data and Information Cleaning and Preprocessing. pp. 22–30.

Zhou, Y., Yang, Y., Lu, H., Chen, L., Li, Y., Zhao, Y., Qian, J., Xu, B., 2018. How far we have progressed in the journey? an examination of cross-project defect prediction. ACM Trans. Softw. Eng. Methodol. (TOSEM) 27 (1), 1.

Zimmermann, T., Nagappan, N., 2008. Predicting defects using network analysis on dependency graphs. In: Proceedings of the 30th International Conference on Software Engineering. ICSE, IEEE, pp. 531–540.

Zhou Xu received the B.S. degree from Huazhong Agricultural University, China, in 2014 and the Ph.D. degree from Wuhan University, China, in 2019. He is now an Assistant Professor at the School of Big Data and Software Engineering, Chongqing University, China. His research interests include software defect prediction, feature engineering, and machine learning.

Li Li is a Lecturer in the Faculty of Information Technology, Monash University. Prior to joining Monash, he was a research associate in Software Engineering at the University of Luxembourg and an honorary research associate at University College London. He obtained his Ph.D. degree in November 2016 from the University of Luxembourg. His research interests are in the areas of Android Security, Static Analysis, Machine Learning, Deep Learning, and Empirical Study.

Meng Yan is now an Assistant Professor at the School of Big Data and Software Engineering, Chongqing University, China. Prior to joining Chongqing University, he was a Postdoc at Zhejiang University advised by Prof.Shanping Li and Dr. Xin Xia. he got his Ph.D degree in June 2017 under the supervision of Prof. Xiaohong Zhang from Chongqing University, China. H is currently research focuses on how to improve developers' productivity, how to improve software quality and how to reduce the effort during software development by analyzing rich software repository data.

Jin Liu received the Ph.D. degree in computer science from the State Key Lab of Software Engineering, Wuhan University, China, in 2005. He is currently a Professor in the School of Computer Science, Wuhan University. His research interests include software engineering, machine learning, and interactive collaboration on the Web.

Xiapu Luo received the Ph.D. degree in computer science from The Hong Kong Polytechnic University in 2007, and was a Post-Doctoral Research Fellow with the Georgia Institute of Technology. Now, he is an Associate Professor with the Department of Computing and an Associate Researcher with the Shenzhen Research Institute, The Hong Kong Polytechnic University. His current research focuses on smart phone security and privacy, network security and privacy, and Internet measurement.

John Grundy is the Senior Deputy Dean for the Faculty of Information Technology and a Professor of Software Engineering at Monash University. He hold the BSc(Hons),M.Sc. and Ph.D. degrees, all in Computer Science, from the University of Auckland. He is a Fellow of Automated Software Engineering, Fellow of Engineers Australia, Certified Professional Engineer, Engineering Executive, Member of the ACM and Senior Member of the IEEE.

Yifeng Zhang received his master degree from Wuhan University, China, in 2019. His research interest focuses on software engineering and machine learning.

Xiaohong Zhang received the M.S. degree in applied mathematics and the Ph.D. degree in computer software and theory from Chongqing University, China, in 2006. He is currently a Professor and the Vice Dean of the School of Big Data and Software Engineering, Chongqing University. His current research interests include data mining of software engineering, topic modeling, image semantic analysis, and video analysis.