11-2017

File-level defect prediction: Unsupervised vs. supervised models

Meng YAN

Yicheng FANG

David LO
Singapore Management University, davidlo@smu.edu.sg

Xin XIA

Xiaohong ZHANG

DOI: https://doi.org/10.1109/ESEM.2017.48

File-Level Defect Prediction: Unsupervised vs. Supervised Models

Meng Yan*†, Yicheng Fang†, David Lo‡, Xin Xia§✓, and Xiaohong Zhang†✓
*College of Computer Science and Technology, Zhejiang University, Hangzhou, China
†School of Software Engineering, Chongqing University, Chongqing, China
‡School of Information Systems, Singapore Management University, Singapore
§Department of Computer Science, University of British Columbia, Canada
Email: {meng.yan, fangyicheng, xhongz}@cqu.edu.cn, davidlo@smu.edu.sg, xxia02@cs.ubc.ca

***Abstract*—*Background:* Software defect models can help software quality assurance teams to allocate testing or code review resources. A variety of techniques have been used to build defect prediction models, including supervised and unsupervised methods. Recently, Yang et al. [1] surprisingly find that unsupervised models can perform statistically significantly better than supervised models in effort-aware change-level defect prediction. However, little is known about relative performance of unsupervised and supervised models for effort-aware file-level defect prediction. *Goal:* Inspired by their work, we aim to investigate whether a similar finding holds in effort-aware file-level defect prediction. *Method:* We replicate Yang et al.'s study on PROMISE dataset with totally ten projects. We compare the effectiveness of unsupervised and supervised prediction models for effort-aware file-level defect prediction. *Results:* We find that the conclusion of Yang et al. [1] does not hold under within-project but holds under cross-project setting for file-level defect prediction. In addition, following the recommendations given by the best unsupervised model, developers needs to inspect statistically significantly more files than that of supervised models considering the same inspection effort (i.e., LOC). *Conclusions:* (a) Unsupervised models do not perform statistically significantly better than state-of-art supervised model under within-project setting, (b) Unsupervised models can perform statistically significantly better than state-of-art supervised model under cross-project setting, (c) We suggest that not only LOC but also number of files needed to be inspected should be considered when evaluating effort-aware file-level defect prediction models.**

***Index Terms*—Replication Study, Inspection Effort, Effort-aware Defect Prediction**

I. INTRODUCTION

Fixing defects is a vital activity during software maintenance. It can cost up to 80% of software development budget [2]. To help developers better manage defects, a number of software engineering studies have proposed defect prediction models. A defect prediction model can provide a list of likely buggy software artifacts (e.g., modules or files) in an early stage. As a result, software quality assurance (SQA) teams can use the list to allocate resources effectively by focusing on the likely buggy parts [3]–[7].

Researchers have proposed various defect prediction models [6], [8]–[21]. Most of them are based on supervised learning from labeled training data. However, the cost of collecting training data is a barrier of adopting defect prediction in industry [22], [23]. For new projects or projects with limited development history, there is often not enough defect information for building a model. This is the main challenge of supervised defect prediction. An alternative solution is the cross-project defect prediction (CPDP) which uses labeled training data from other projects [23]–[25]. However, the main challenge of CPDP is the heterogeneity of projects; this heterogeneity makes it hard to learn a model from a project and use it for another [24], [26]–[28].

To address such challenge, unsupervised models can be used [13], [24], [29]. The main advantage of unsupervised defect prediction is that it does not require access to training data. However, there is only a few studies in the literature focusing on unsupervised defect prediction. One reason is that unsupervised prediction models usually underperform supervised ones in terms of prediction performance [24].

Both supervised and unsupervised prediction models will provide a list of likely buggy code units which developers can focus on during subsequent SQA activities. However, due to limited resources for code inspection and testing, it is expensive and impractical to inspect all potential defective code units. Therefore, effort-aware defect prediction models are proposed [30]–[33]. It aims to reduce the candidate set of code units to inspect by ranking code units based on predicted defect density. In this way, it could find more defects per unit code inspection effort.

Recently, Yang et al. [1] surprisingly find that unsupervised models can perform statistically significantly better than supervised models for effort-aware change-level defect prediction. Fu and Menzies [34] and Huang et al. [35] revisit and refute their findings for change-level defect prediction. However, to the best of our knowledge, little is known about relative performance of unsupervised and supervised models for effort-aware **file-level** defect prediction. Therefore, to fill this gap, we set out to revisit the findings of Yang et al. [1] for effort-aware file-level defect prediction by conducting an empirical study considering both within-project and cross-project validation settings. The difference between change-level and file-level defect prediction lies on the development phase when they are employed. Change-level defect prediction is conducted when each change is submitted. It aims to be a continuous activity of quality control. File-level defect prediction is usually conduct-

✓ Corresponding authors.

ed before a product release. It aims to be a quality control step before a release. They can complement each other to improve the quality of the upcoming release.

In summary, the main contributions of this paper are as follows:

- We conduct an empirical study of effort-aware file-level defect prediction by comparing various supervised and unsupervised defect prediction models on ten public projects. In addition, we consider both within-project and cross-project prediction settings.
- Our study highlights that the conclusion of Yang et al. [1] does not hold under within-project but holds under cross-project setting for file-level defect prediction.
- We investigate number of files needed to be inspected using unsupervised and supervised models considering the same number of lines of code to inspect. The results show that following recommendations given by the best unsupervised model developers need to inspect more files given the same amount of lines of code (LOC). This finding suggests that not only LOC but also number of files needed to be inspected should be considered as effort for evaluating effort-aware models.

Paper Organization. The rest of this paper is structured as follows. Section 2 presents the background and related work of our study. Section 3 presents the different prediction models that are investigated in this work. In Section 4, we describe the study setup. Section 5 presents experiment results and their analysis. In Section 6, we summarize the main threats of our study. At last, in Section 7, we conclude and present future plans.

II. Background and Related Work

In this section, we first introduce supervised defect prediction models. Then, we briefly review the existing work on unsupervised defect prediction models. Finally, we describe existing work on effort-aware defect prediction.

A. Supervised Defect Prediction

Supervised defect prediction models are built on the historical labelled source code files (i.e., labeled as either buggy or clean). Various software metrics are extracted from source code files, e.g., lines of code, number of methods, number of attributes. The defect information of labeled files is usually collected through various data sources, such as version control system and issue tracking system. Various machine learning techniques can be used to build supervised defect prediction models, e.g., Logistic regression, Decision tree, Naive Bayes and Random Forest [8], [26], [36], [37].

There are two kinds of prediction settings in supervised defect prediction, namely within-project defect prediction (WPDP) and cross-project defect prediction (CPDP). In WPDP, the whole process is performed within a single project. The model is built by learning from labeled instances of the project. And the model is adopted to predict the labels of unknown files within the project. The limitation of WPDP is that it is difficult to build accurate models on new projects or projects with limited historical data [38]. In addition, collecting defect information involves substantial effort which may prevent its adoption in industry [22], [23]. In CPDP, the model is built on the labeled instances from one or several projects (i.e., source projects). And the model is applied to predict the labels of unknown files of another project (i.e., target project). The major limitation of CPDP is the distribution of defects in the source projects and target projects are different, which may lead to low accuracy [24], [27].

B. Unsupervised Defect Prediction

Despite the promising results achieved by supervised models, the expensive effort cost of collecting defect data is still a barrier for applying it in practice. The advantage of unsupervised model is that it does not need any labelled data. A typical process of unsupervised defect prediction consists of two steps: first, files are grouped into k clusters (usually two clusters); second, each cluster is labeled as buggy or clean [24]. There exists a limited number of studies on unsupervised defect prediction [13], [24], [29], [38], [39]. One reason is that the unsupervised models usually do not perform as well as supervised ones [24].

An early attempt of unsupervised defect prediction is made by Zhong et al. [29]. They propose to apply k-means and neural-gas clustering techniques [40] in defect prediction. They find that neural-gas is better than k-means. One issue of their work is that the choice of the number of clusters and the labeling step which requires manual effort, and it is hard to determine a good value in practice. Bishnu and Bhattacherjee [29] propose to apply quad trees to initialize cluster centers in k-means based unsupervised defect prediction. Yang et al. [13] propose to adopt an affinity propagation clustering algorithm to predict defective files. Nam and Kim [38] propose to use thresholds of selected metrics to label the clusters. Zhang et al. [24] adopt spectral clustering to build a connectivity-based unsupervised prediction model.

C. Effort-Aware Defect Prediction

The objective of defect prediction is to support software quality assurance (SQA) activities, such as unit testing and code review. Test managers and quality managers allocate more efforts to test or review software entities which are predicted as buggy. However, due to limitation of SQA resources, it is impractical to inspect all files that are predicted as buggy. In practice, it is important to consider the effort of applying SQA activities after defect prediction. Thus, effort-aware defect prediction models have been proposed [30]–[33]. Many of the traditional classification algorithms (e.g., logistic regression, decision trees and support vector machines) which perform well in terms of traditional performance measures are found to perform poorly when assessed considering effort involved in performing inspection activities [33], [41].

Mende et al. [31] include the notion of effort into a defect prediction model. Kamei et al. [30] revisit whether common findings in traditional defect prediction still hold for effort-aware prediction. Koru et al. [42], [43] suggest that smaller modules should be inspected with higher priority, since more defects would be detected per unit code inspection effort. Based on this finding, Menzies et al. [44] name the finding of Koru et al. as "ManualUp" model. As a result, they find that the ManualUp model performs well in effort-aware prediction performance. Shihab et al. [32] investigate the different choices of measures (e.g., LOC) used as proxy of effort in defect prediction. Mezouar et al. [33] compare local and global effort-aware defect prediction models.

Recently, Yang et al. [1] find unsupervised models can perform better than supervised models for effort-aware change-level defect prediction. Fu and Menzies [34] and Huang et al. [35] revisit and refute their findings for change-level defect prediction. Although both supervised modes [30], [31], [33] and unsupervised models [43], [44] have been investigated for effort-aware file-level defect prediction, little is known about their relative performance using the same dataset. To fill this gap, we perform an empirical study by replicating the work of Yang et al. [1]. The main difference between this work and Yang et al's work [1], Fu and Menzies's work [34] and Huang et al.'s work [35] is: they focus on change-level defect prediction and we focus on file-level defect prediction.

III. File-Level Defect Prediction Methods

In this section, we introduce various effort-aware file-level defect prediction methods which are investigated in this work. We first provide detail description of unsupervised models. Subsequently, we describe a collection of state-of-art supervised models.

A. Unsupervised Models

The objective of a defect prediction model is to determine risky code for further software quality assurance activities. In terms of the model output, there are two kinds of outcomes. One is a classification outcome (i.e., classify each entity as defective or clean). The other is a ranking result based on defect-prone risk value, which can provide a list of entities (i.e., files or modules) ordered by the risk. The latter is more suitable for defect prediction purpose [31], [45], since the SQA team can focus on the highly-ranked files (i.e., with higher defect-prone risk values) for further SQA activities (e.g., code review and unit tests).

In our study, we use the unsupervised models proposed by Yang et al. [1]. The intuition of the unsupervised models is based on the finding by Koru et al. [42], [43]: they found that "smaller modules are proportionally more defect-prone and should be inspected first, as more defects would be detected per unit code inspection effort". The unsupervised models use software metrics; we list them in Table I. Following Yang et al.'s study [1], we exclude lines of code (which is an effort proxy metric) for building unsupervised models. In modeling step, for each metric, each unsupervised model is built by ranking files in descending order of the reciprocal of the metric value. Formally, let $R(i)$ represents the risk value of file i, and $M(i)$ represents the metric value of file i. The relationship between the risk and metric value is given by: $R(i) = 1/M(i)$. In this way, files with smaller product metric values will be ranked higher. In the latter part of the paper, we use the term unsupervised models to refer to Yang et al.'s unsupervised models presented above.

B. Supervised Models

To compare effectiveness between unsupervised and supervised models, we investigate a collection of state-of-art supervised techniques. In detail, six families with a total of 31 techniques are used in our work. These supervised models are selected due to the following reasons. First, these supervised models are commonly used in defect prediction studies [8], [36], [37], [46], [47]. Second, all of them are investigated in Yang et al.'s work [1], and most of them (except Random Forest) are revisited in Ghotra et al.'s work [36].

Table II summaries the 31 supervised models, which are grouped into six families, namely Function, Lazy, Rule, Bayes, Tree and Ensemble. In the Function family, we use regression models, neural networks and support vector machine, including EALR [48] (i.e., Effort-Aware Linear Regression), Simple Logistic (SL), Radial Basis Functions Network (RBFNet), and Sequential Minimal Optimization (SMO). The Lazy family represents lazy learning methods, and we use the K-Nearest Neighbour (IBk) method. The Rule family represents models based on rules, including propositional rule (JRip) and ripple down rules (Ridor). Bayes family represents probability-based models, and we investigate the most popular one, namely Naive Bayes (NB). The Tree family represents decision tree based methods, including J48, Logistic Model Tree (LMT) and Random Forest (RF). In the last family, we have four ensemble learning methods: Bagging, Adaboost, Rotation Forest and Rotation Subspace. Different from other models, ensemble learning models are built with multiple base classifiers. These supervised models take as input some parameters. We use the same parameter settings employed in Yang et al.'s work [1].

IV. Study Setup

In this section, we introduce the experimental setup. First, we provide descriptions of datasets used in our study. Second, we present three research questions that we would like to investigate. Third, we introduce validation methods, evaluation measures, and statistical tests.

A. Dataset

In this work, we use a publicly available dataset with a total of ten projects in PROMISE [49]. These projects have been widely used in defect prediction studies [24], [33]. Table III shows descriptive statistics of the dataset.

B. Research Questions

In this paper, we are interested to investigate the following three research questions:

- RQ1: How well can unsupervised models perform as compared to supervised ones under within-project setting?

TABLE I: List of metrics

Category	Product Metric
Complexity	Lines of Code (LOC)
	Weighted Methods per Class (WMC)
	Number of Public Methods (NPM)
	Average Method Complexity (AMC)
	Max McCabe's Cyclomatic Complexity (Max_cc)
	Avg McCabe's Cyclomatic Complexity (Avg_cc)
	Measure of Aggregation (MOA)
Coupling	Coupling between object classes (CBO)
	Response of a Class (RFC)
	Afferent Couplings (CA)
	Efferent Couplings (CE)
	Inheritance Coupling (IC)
	Coupling Between Methods (CBM)
Cohesion	Lack of cohesion in methods (LCOM)
	Lack of cohesion in methods (LCOM3)
	Cohesion Among Methods of Class (CAM)
Abstraction	Depth of Inheritance Tree (DIT)
	Number Of Children (NOC)
	Measure of Functional Abstraction (MFA)
Encapsulation	Data Access Metric (DAM)

TABLE II: Summary of supervised models

Family	Model	Abbreviation
Function	Linear Regression	EALR
	Simple Logistic	SL
	Radial basis functions network	RBFNet
	Sequential Minimal Optimization	SMO
Lazy	K-Nearest Neighbour	IBk
Rule	Propositional rule	JRip
	Ripple down rules	Ridor
Bayes	Naive Bayes	NB
Tree	J48	J48
Ensemble	Logistic Model Tree	LMT
	Random Forest	RF
	Bagging	BG+LMT, BG+NB, BG+SL, BG+SMO, and BG+J48
	Adaboost	AB+LMT, AB+NB, AB+SL, AB+SMO, and AB+J48
	Rotation Forest	RF+LMT, RF+NB, RF+SL, RF+SMO, and RF+J48
	Random subspace	RS+LMT, RS+NB, RS+SL, RS+SMO, and RS+J48

TABLE III: Descriptive statistics of our dataset

Dataset	Project	#Files	#Defects	%Defects
PROMISE	Ant v1.7	745	166	22.3%
	Camel v1.6	965	188	19.5%
	Ivy v1.4	241	16	6.6%
	Jedit v4.0	306	75	24.5%
	Log4j v1.0	135	34	25.2%
	Velocity v1.6	229	78	34.1%
	POI v2.0	314	37	11.8%
	Tomcat v6.0	858	77	9.0%
	Xalan v2.4	723	110	15.2%
	Xerces v1.3	453	69	15.2%

- RQ2: How well can unsupervised models perform as compared to supervised ones under cross-project setting?
- RQ3: How many files need to be inspected using unsupervised and supervised models considering the same amount of LOC to inspect?

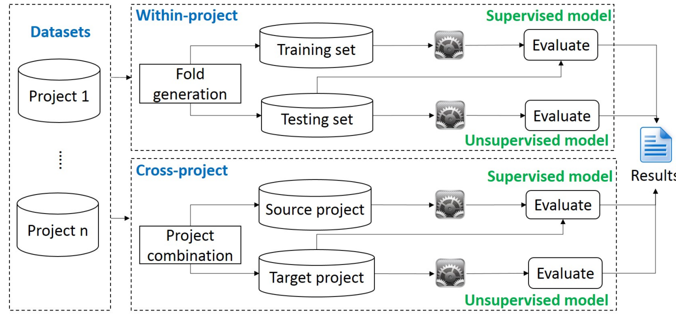


Fig. 1: Overview of validation methods

RQ1 and RQ2 are designed to compare the effectiveness of unsupervised and supervised models in terms of two different prediction settings. Although unsupervised effort-aware file-level models have been proposed, little is known about the answers of these two research questions; no prior work have compared the two families of techniques using the same dataset in effort-aware file-level defect prediction. This work will perform an in-depth empirical study to answer the two questions. In addition, the objective of the effort-aware model is to find more defects by using less inspection cost (measured in terms of LOC inspected). However, existing approaches ignore number of files needed to be inspected. Inspecting many files may cost more due to context switching involved. RQ3 is designed to investigate the number of files needed to be inspected in unsupervised and supervised models considering the same amount of LOC to inspect.

C. Settings Considered

To compare the effectiveness between unsupervised and supervised methods, we consider two settings: within-project (in RQ1) and cross-project (in RQ2) validation as Figure 1 shows.

For within-project validation, we use 10 times 10-fold cross-validation. In detail, we repeat each experiment 10 times within a project. In each cross-validation, we randomly divide data from each project into 10 sub-samples of approximately equal size, each sub-sample is used once as the testing data and the remaining data is used for training. In this way, we obtain 100 effectiveness values in each project for each supervised model. For each unsupervised model, we also apply the 10 times 10-fold cross-validation setting. This enables unsupervised models use the same testing data each time to make a fair comparison.

For cross-project validation, we build a supervised model by training on one project (source project) and testing on another project (target project). For a dataset with n projects, there will be $n*(n-1)$ $\langle$source, target$\rangle$ combinations. In addition, in order to perform a fair comparison, when evaluating each unsupervised model against a supervised model, we keep the target project identical with the supervised model in each step of the comparison process.

D. Evaluation Measures

Following prior studies [31], [33], we use LOC as proxy measure of effort (aka. cost) involved in inspecting a file.

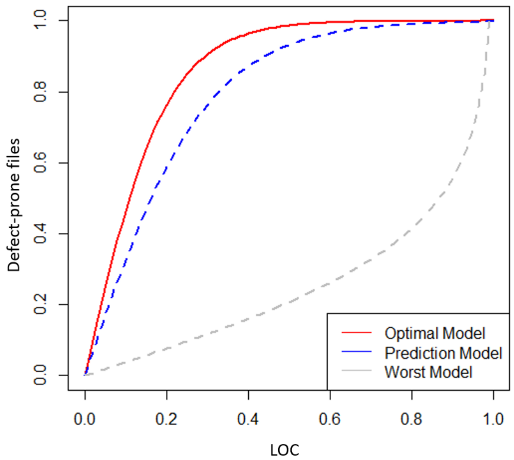


Fig. 2: Illustration of the LOC-based Alberg diagram

Two typical effort-aware evaluation measures are used in our work, namely ACC and P_{opt} [1], [33], [48]. ACC denotes the proportion of defective files identified when developers inspect the top ranked files until 20% of the total LOC is inspected. P_{opt} is the normalized version of effort-aware performance indicator originally proposed by Mende and Koschke [31].

In detail, P_{opt} is based on the area under the effort curve in an Alberg diagram [1]. Figure 2 shows an example of an LOC-based Alberg diagram [1]. There are three curves corresponding to a target, prediction model m, the "optimal" model and the "worst" model. The opt represents the term "optimal". Using the optimal model, all the files are ranked in decreasing actual defect density; let $Area(optimal)$ represents the area under the optimal curve. Using the predicted model, all the files are ranked in order of predicted risk value; let $Area(m)$ represents the area under the predicted curve. Using the worst model, all the files are ranked in ascending order of its defect density; let $Area(worst)$ represents the area under the worst curve. Following [1], [48], $P_{opt}(m)$ is computed as follows:

$$P_{opt}(m) = 1 - \frac{Area(optimal) - Area(m)}{Area(optimal) - Area(worst)} \quad (1)$$

E. Statistical Methods

When comparing supervised and unsupervised models, we use two kinds of statistical tests to examine whether the models' effectiveness are significantly different. To examine the performance difference in view of the global ranking of all the models, we use the Scott-Knott test. To examine the performance difference in view of each comparison between unsupervised model and supervised model, we use the Wilcoxon signed-rank test. We briefly introduce the two tests in this section.

Following prior studies [1], [36], we use Scott-Knott test [50] to group all the models into statistically distinct ranks in both RQ1 and RQ2. This test is used to examine whether some models outperform others and create a global ranking of models. In detail, Scott-Knott test performs the grouping process in a recursive way. At first, Scott-Knott test uses hierarchical cluster analysis to partition all the models into two ranks based on the mean performance. After that, if the divided ranks are significantly different, then Scott-Knott test recursively executes again within each rank to further divide the ranks. In this way, the test will terminate when ranks can no longer be divided into statistically distinct ranks.

We use Wilcoxon signed-rank test to examine whether the performance difference between two models are statistically significant in RQ1 and RQ2. We also use the Benjamini-Hochberg (BH) procedure to adjust p-values since we perform multiple comparisons [51]. After that, if the test shows a significant difference, we compute Cliff's delta which is a non-parametric effect size measure to examine whether the magnitude of the difference is substantial or not [1], [52]. Based on the value of delta, the difference can be considered trivial ($|\delta| < 0.147$), small ($0.147 \leqslant |\delta| < 0.33$), moderate ($0.33 \leqslant |\delta| < 0.474$), or large ($\geqslant 0.474$) [1], [52].

V. Experimental Results

In this section, we present empirical results and their implications for the three research questions.

A. RQ1: Unsupervised vs. Supervised (Within Project)

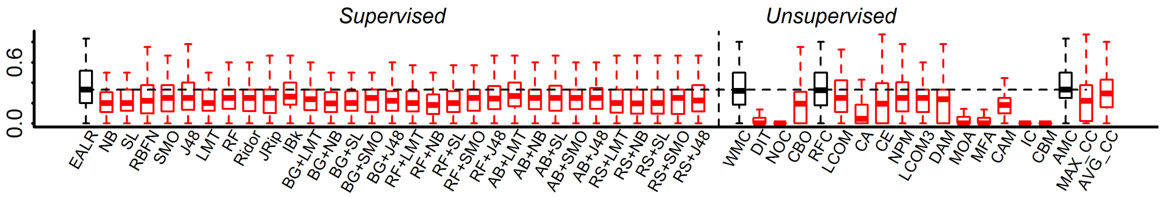
Visualizations. Following the visualization technique used in Yang et al.'s work [1], we use box-plots with different colors to present the distribution difference of effectiveness values between unsupervised and supervised models. Each box-plot presents the median (the horizontal line within the box), the 25th percentile (the lower side of the box), and the 75th percentile (the upper side of the box). In order to show the statistical difference between supervised models and unsupervised models, the box-plots consists of three colors, namely blue, red and black. In detail, we set the best supervised model (in terms of the median value) as the baseline and we use a horizontal dotted line to represent the median performance of the best supervised model. The different colors then carry the following meanings:

A blue color box-plot represents that the corresponding model outperforms the best supervised model with a statistical significance according to the Wilcoxon signed-rank test where the Benjamini-Hochberg (BH) corrected *p-value* is less than 0.05. Additionally, the magnitude of the performance difference between the two models is not trivial according to Cliff's delta, i.e., $|\delta| \geqslant 0.147$.

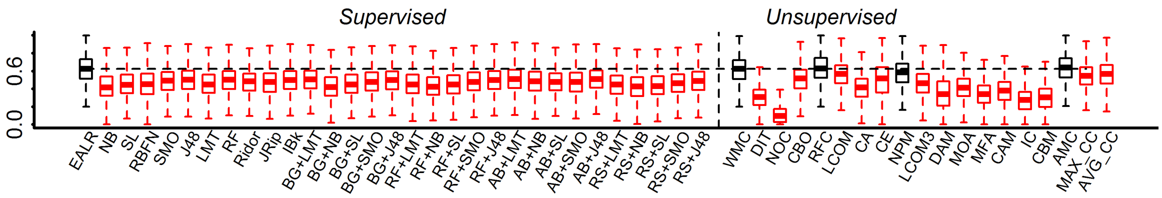
A red color box-plot represents that the corresponding model performs significantly worse than the best supervised model according to the Wilcoxon signed-rank test. Additionally, the magnitude of the performance difference between the two models is not trivial according to Cliff's delta.

A black color box-plot represents that the difference between the corresponding model and the best supervised model is not significant or the magnitude of the difference is trivial according to Cliff's delta, i.e., $|\delta| < 0.147$.

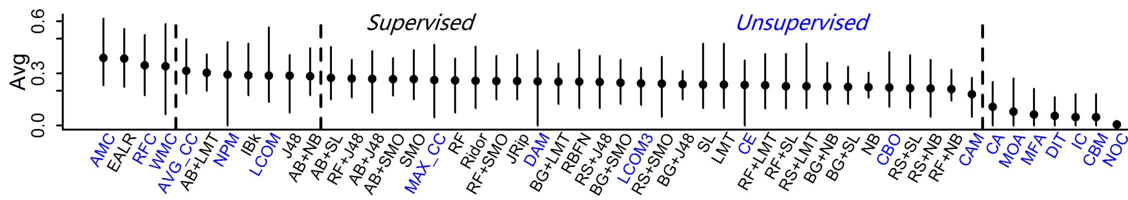
We also draw two diagrams presenting results of running Scott-Knott tests for P_{opt} and ACC respectively. Each diagram shows global ranking of models across projects in terms of their P_{opt} and ACC scores respectively.



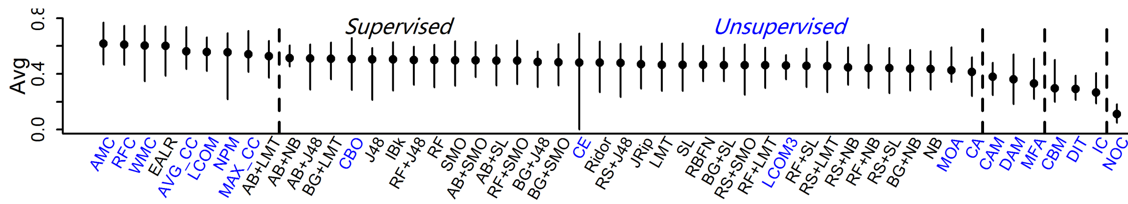
(a) ACC results



(b) P_{opt} results



(c) Scott-Knott test results for ACC



(d) Scott-Knott test results for P_{opt}

Fig. 3: Performance comparison under within-project setting

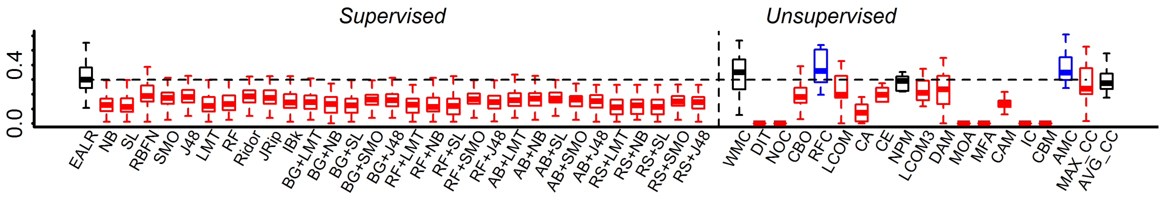
Results. Figure 3 shows the various visualizations highlighting the overall results of PROMISE dataset considering 10 times 10-fold cross-validation. We draw two box plots for ACC and P_{opt} respectively as Figure 3(a) and 3(b). In addition, we draw the Scott-Knott test results for ACC and P_{opt} as Figure 3(c) and 3(d). In Figure 3(c) and (d), the y-axis represents the average performance. The blue labels indicate unsupervised models, while the black labels indicate supervised ones. The dotted lines represent groups of equivalent models as output by the Scott-Knott test.

From Figure 3, we observe the following findings: first, the best supervised model is EALR in terms of both ACC and P_{opt}. It outperforms the other supervised models significantly. Second, there is no unsupervised model that outperforms EALR significantly, as there is no blue box. In addition, when compared with EALR, three unsupervised models achieve similar performance in terms of ACC, while four unsupervised models achieve similar performance in terms of P_{opt}. Third, according to the Scott-Knott test, EALR is among members of the first group in terms of both ACC and P_{opt}.

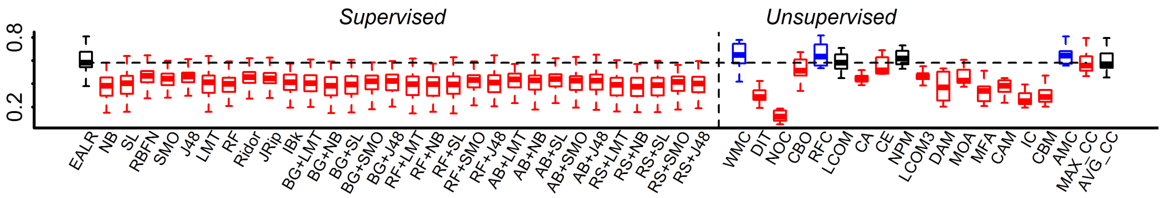
Table IV lists the median ACC and P_{opt} for the best supervised model (i.e., EALR) and the best two unsupervised models (i.e., AMC and RFC) for each project. We use "√" and "×" to indicate whether the unsupervised model performs significantly "better" and "worse" than the best supervised model according to the Wilcoxon's signed rank test (after Benjamini-Hochberg correction) respectively. AVG represents the average performance over the ten projects. The row "W/T/L" represents the number of projects for which the unsupervised model performs better, equally well or worse than the best supervised model. In addition, if there is a significant difference, a "light gray", "gray", "deep gray" and

TABLE IV: Within-project validation results for each project: best supervised model vs. best two unsupervised models

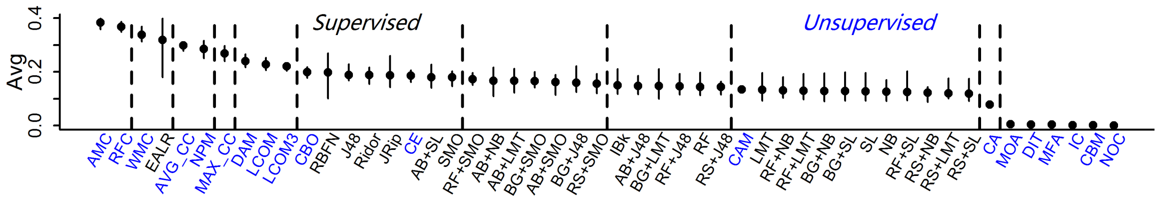
Project	ACC			Popt		
	EALR	AMC	RFC	EALR	AMC	RFC
Ant	0.211	0.235√	0.205	0.541	0.550	0.556
Camel	0.462	0.400×	0.500√	0.728	0.666×	0.740√
Ivy	0.250	0.000	0.000	0.475	0.482	0.570√
Jedit	0.286	0.444√	0.369√	0.584	0.653√	0.671√
Log4j	0.333	0.414	0.250×	0.638	0.668	0.535×
Velocity	0.500	0.500	0.500	0.750	0.755√	0.746
POI	0.333	0.250×	0.354	0.569	0.536	0.611
Tomcat	0.250	0.286	0.222×	0.551	0.581√	0.524×
Xalan	0.304	0.308	0.250×	0.614	0.619	0.599
Xerces	0.477	0.600√	0.586√	0.711	0.775√	0.765√
AVG	0.341	0.344	0.324	0.616	0.629	0.632
W/T/L		3/5/2	3/4/3		4/5/1	4/4/2



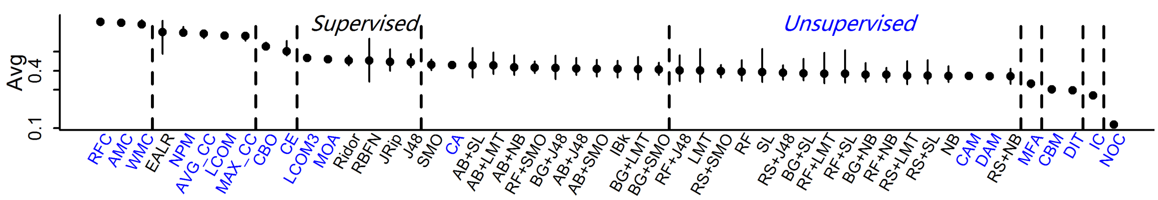
(a) ACC results



(b) P_{opt} results



(c) Scott-Knott test results for ACC



(d) Scott-Knott test results for P_{opt}

Fig. 4: Performance comparison under cross-project setting

"yellow" background indicate "trival", "small", "moderate" and "large" magnitude of difference according to Cliff's delta respectively.

From Table IV, we have the following findings: first, there is no significant difference in terms of AVG between the best supervised model and two unsupervised models. Second, in most projects, there is no significant difference between the best supervised model and two unsupervised models. In some projects, the best unsupervised model significantly outperforms the best supervised model (e.g., 3 projects in AMC and RFC in terms of ACC). Also, the best supervised model can also significantly outperform the best two unsupervised models in some projects (e.g., 2 projects in AMC and 3 projects in RFC in terms of ACC). Third, considering the significant cases, we only observe two moderate difference and no large difference. Most of the magnitudes of the differences (20/22) are "trival" or "small" according to the Cliff's delta .

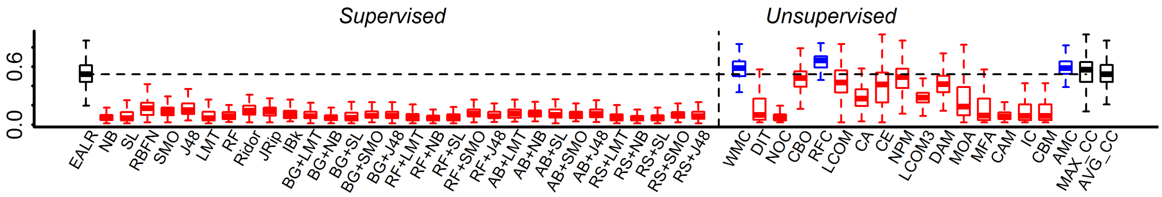
Implication. The above findings suggest that the conclusion of Yang et al.'s study [1] does not hold under within-project for effort-aware file-level defect prediction. Namely, unsupervised models do not perform statistically significantly better than state-of-art supervised model under within-project for effort-aware file-level defect prediction.

Unsupervised models do not perform statistically significantly better than state-of-art supervised model for effort-aware file-level defect prediction, considering within-project evaluation setting.

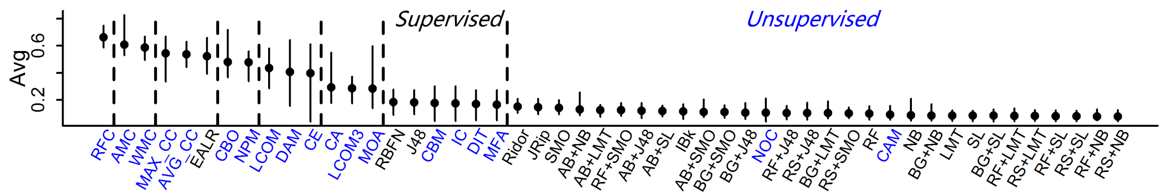
B. RQ2: Unsupervised vs. Supervised (Cross Project)

Results. Figure 4 presents the cross-project validation results. Following RQ1, we use box-plot to visualize the results. Since there are ten projects in our experiment, there are $10 * 9 = 90$ $\langle$source,target$\rangle$ combinations for each prediction model. As a result, each box-plot has 90 effectiveness values.

From Figure 4, we have the following findings: first, the best supervised model is EALR in terms of both ACC and P_{opt}. Second, in terms of ACC, there are two unsupervised models (i.e., RFC and AMC) which significantly outperform the best supervised model EALR. In terms of P_{opt}, there are three unsupervised models (i.e., WMC, RFC and AMC) which significantly outperform the best supervised model. Third, according to the Scott-Knott test results, the models in the first group are unsupervised models and the best supervised model EALR is among members of the second group in terms of both ACC and P_{opt}.

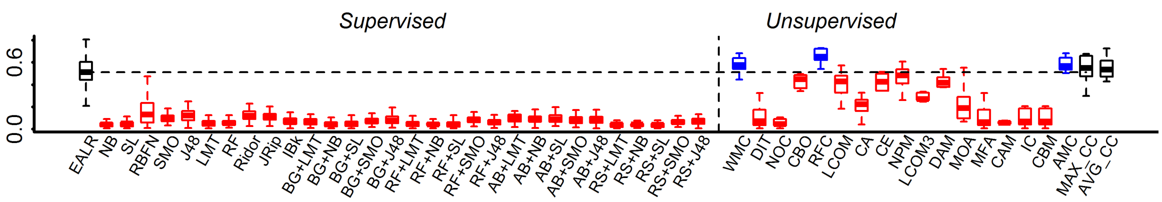


(a) Proportion of files to be inspected

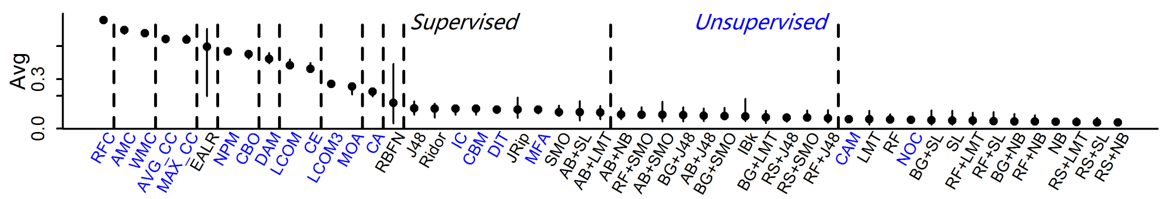


(b) Scott-Knott test for proportion of files to be inspected

Fig. 5: Proportion of files to be inspected under within-project setting



(a) Proportion of files to be inspected



(b) Scott-Knott test for proportion of files to be inspected

Fig. 6: Proportion of files to be inspected under cross-project setting

Implication. The above findings suggest that the conclusion of Yang et al. [1] holds under cross-project for effort-aware file-level defect prediction. Namely, unsupervised models can perform statistically significantly better than state-of-art supervised model under cross-project for effort-aware file-level defect prediction.

> *Unsupervised models can perform statistically significantly better than state-of-art supervised model for effort-aware file-level defect prediction, considering cross-project evaluation setting.*

C. RQ3: Analysis on the number of files for inspection

Results. We obtain the proportion of files needed to be inspected when computing the ACC value (i.e., when 20% of total LOC is inspected). Figure 5 and 6 present the proportion of files needed to be inspected using different models under within-project and cross-project setting respectively.

From Figure 5 and 6, we have the following findings: first, the proportion of files to inspect for EALR is significantly higher than the other supervised models. Second, the proportion of files to inspect for the best three unsupervised models (i.e., AMC, RFC and WMC) are significantly higher than the best supervised model (i.e., EALR) under within-project and cross-project setting, as the corresponding boxes are blue. Third, according to the Scott-Knott test result, the first group and second group are unsupervised models, the EALR is located in the third and fourth group under within-project and cross-project setting respectively.

Moreover, Table V lists the median proportion of files to inspect in each project for the best supervised model and the best two unsupervised models under within-project validation. From Table V, we have the following findings: first, the average proportion of files to inspect for the unsupervised models AMC and RFC exceed that of EALR by 14.3% and 24.1% respectively. Second, the proportion of files to inspect for AMC is significantly higher than that of EALR in four projects, and there is no significant difference in the remaining six projects according to the Wilcoxon's signed-rank test (after Benjamini-Hochberg correction). Meanwhile, the proportion of files to inspect for RFC is significantly higher than that of EALR in nine projects. Third, considering the significant cases, twelve of them have a *large* magnitude of difference according to the Cliff's delta.

Implication. The above findings suggest that following rec-

TABLE V: Proportion of files needed to be inspected for each project under within-project validation

Project	EALR	AMC	RFC
Ant	0.561	0.547	0.607√
Camel	0.518	0.521	0.653√
Ivy	0.625	0.653	0.729√
Jedit	0.524	0.525	0.722√
Log4j	0.407	0.555√	0.593√
Velocity	0.609	0.609	0.696
POI	0.422	0.572√	0.581√
Tomcat	0.535	0.628√	0.680√
Xalan	0.521	0.558	0.639√
Xerces	0.511	0.813√	0.767√
AVG	0.523	0.598	0.667
W/T/L		0/6/4	0/1/9

ommendations given by the best unsupervised model, developers need to inspect statistically significantly more files than supervised models when the same amount of effort (measured in terms of LOC) is spent. In addition, although EALR outperforms other supervised models considering ACC and P_{opt}, it also needs to inspect more files than other supervised models. This highlights a problem of evaluating the performance of effort-aware models by only using LOC as the effort measure. The reason is that the effort needed for inspecting the same number of LOC spread across different number of files are likely to differ due to context switching cost. If a developer needs to inspect multiple files, it requires him/her to context switch between the files. The context switching cost can be large especially when a large number of files need to be inspected. Thus, we suggest that not only LOC but also number of files needed to be inspected should be considered when evaluating effort-aware file-level defect prediction models.

By following recommendations given by the best unsupervised model, developers need to inspect more files given the same amount of lines of code (LOC). Not only LOC but also number of files needed to be inspected should be considered when evaluating effort-aware file-level defect prediction models.

VI. Threats to Validity

In this section, we discuss potential aspects which may threat the validity of our study.

Internal Validity. One main threat to internal validity of our study is the setting of the cut-off value when calculating ACC. We set the cut-off value as 0.2 by following prior works [1], [48]. To reduce this potential threat, we use an additional performance measure (i.e., P_{opt}) and analyze the number of files to be inspected to evaluate the models.

External Validity. Threats to external validity relate to generalizability of our results. First, our study uses lines of code as measure of effort by following similar prior work [31], [33]. However, one work [32] found that the complexity measures also have a strong correlation with effort. Using other measures of efforts such as McCabe cyclomatic complexity to replicate our study can be useful in generalizing our findings. Second, projects investigated in our experiments all come from PROMISE public projects repository [49]. To reduce this threat further, as a future work, it will be interesting to replicate our study on more datasets. We have provided the necessary details and scripts that will make it easy for others to replicate our study.

VII. Conclusion and Future work

Recently, Yang et al. [1] found that unsupervised models can perform significantly better than supervised models for effort-aware change-level defect prediction. In this paper, we seek to revisit the finding for effort-aware file-level defect prediction. In detail, we perform an in-depth empirical replication study to investigate whether their findings hold for file-level defect prediction.

In summary, our experimental results from ten public projects show that: (1) the conclusion of Yang et al. [1] does not hold under within-project for effort-aware file-level defect prediction. Namely, unsupervised models do not perform statistically significantly better than state-of-art supervised model under within-project setting. (2) The conclusion of Yang et al. [1] holds under cross-project for effort-aware file-level defect prediction. Namely, unsupervised models can perform statistically significantly better than state-of-art supervised model under cross-project setting. (3) We investigate number of files needed to be inspected given the same amount of LOC. The experimental results show that the proportion of files needed to be inspected in the best unsupervised model are significantly higher than that of supervised models when inspecting same amount of LOC. Meanwhile, the best supervised model EALR also has a significantly higher proportion than that of other supervised models. This finding raises the issue of only regarding LOC as the effort measure when evaluating effort-aware models. We suggest that not only LOC but also number of files needed to be inspected should be considered for evaluating effort-aware file-level defect prediction models.

In the future, in order to investigate the generalization of our findings further, we plan to replicate this work on more datasets, including both open-source and closed-source software projects. In addition, we plan to consider both LOC and number of files needed to be inspected in effort-aware defect prediction models.

Replication package of our study can be downloaded from: *https://bitbucket.org/mengyancqu/esem17defectprediction*

Acknowledgment. We are very grateful to thank Yang et al. [1] for sharing their R scripts and datasets. This work was partially supported by NSFC Program (No. 61602403 and 61572426), Chongqing Research Program of Basic Science & Frontier Technology (No. cstc2017jcyjB0305), and National Key Technology R&D Program of the Ministry of Science and Technology of China (No. 2015BAH17F01).

References

[1] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung, "Effort-aware just-in-time defect prediction: simple unsupervised models

could be better than supervised models," in *FSE*. ACM, 2016, pp. 157–168.
[2] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology, RTI Project*, vol. 7007, no. 011, 2002.
[3] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Applied Soft Computing*, vol. 33, pp. 263–277, 2015.
[4] E. Ceylan, F. O. Kutlubay, and A. B. Bener, "Software defect identification using machine learning techniques," in *32nd EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2006, pp. 240–247.
[5] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 38, no. 2/3, p. 258, 1999.
[6] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *ICSE*. ACM, 2008, pp. 181–190.
[7] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz, "What we have learned about fighting defects," in *Proceedings of the Eighth IEEE Symposium on Software Metrics*. IEEE, 2002, pp. 249–258.
[8] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *TSE*, vol. 33, no. 1, 2007.
[9] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *TSE*, vol. 22, no. 12, pp. 886–894, 1996.
[10] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *TSE*, vol. 22, no. 10, pp. 751–761, 1996.
[11] M. DAmbros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.
[12] A. E. Hassan, "Predicting faults using the complexity of code changes," in *ICSE*. IEEE Computer Society, 2009, pp. 78–88.
[13] B. Yang, Q. Yin, S. Xu, and P. Guo, "Software quality prediction using affinity propagation algorithm," in *IJCNN*. IEEE, 2008, pp. 1891–1896.
[14] T. Lee, J. Nam, D. Han, S. Kim, and H. P. In, "Micro interaction metrics for defect prediction," in *ESEC/FSE*. ACM, 2011, pp. 311–321.
[15] M. Li, H. Zhang, R. Wu, and Z.-H. Zhou, "Sample-based software defect prediction with active and semi-supervised learning," *Automated Software Engineering*, vol. 19, no. 2, pp. 201–230, 2012.
[16] H. Lu and B. Cukic, "An adaptive approach with active learning in software fault prediction," in *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*. ACM, 2012, pp. 79–88.
[17] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *ESEC/FSE*. ACM, 2009, pp. 91–100.
[18] X. Yang, D. Lo, X. Xia, and J. Sun, "Tlel: A two-layer ensemble learning approach for just-in-time defect prediction," *IST*, vol. 87, pp. 206–220, 2017.
[19] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *QRS*. IEEE, 2015, pp. 17–26.
[20] X. Xia, D. Lo, X. Wang, and X. Yang, "Collective personalized change classification with multiobjective search," *IEEE Transactions on Reliability*, vol. 65, no. 4, pp. 1810–1829, 2016.
[21] Y. Zhang, D. Lo, X. Xia, and J. Sun, "An empirical study of classifier combination for cross-project defect prediction," in *COMPSAC*, 2015, pp. 264–269.
[22] A. Tosun, A. Bener, B. Turhan, and T. Menzies, "Practical considerations in deploying statistical methods for defect prediction: A case study within the turkish telecommunications industry," *IST*, vol. 52, no. 11, pp. 1242–1257, 2010.
[23] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.
[24] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *ICSE*. ACM, 2016, pp. 309–320.
[25] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *TSE*, vol. 42, no. 10, pp. 977–998, 2016.
[26] A. E. Camargo Cruz and K. Ochimizu, "Towards logistic regression models for predicting fault-prone code across software projects," in *ESEM*. IEEE Computer Society, 2009, pp. 460–463.
[27] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *ICSE*. IEEE Press, 2013, pp. 382–391.
[28] J. Nam and S. Kim, "Heterogeneous defect prediction," in *ESEC/FSE*. ACM, 2015, pp. 508–519.
[29] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Unsupervised learning for expert-based software quality estimation." in *HASE*. Citeseer, 2004, pp. 149–155.
[30] Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort-aware models," in *ICSM*. IEEE, 2010, pp. 1–10.
[31] T. Mende and R. Koschke, "Effort-aware defect prediction models," in *CSMR*. IEEE, 2010, pp. 107–116.
[32] E. Shihab, Y. Kamei, B. Adams, and A. E. Hassan, "Is lines of code a good measure of effort in effort-aware models?" *IST*, vol. 55, no. 11, pp. 1981–1993, 2013.
[33] M. E. Mezouar, F. Zhang, and Y. Zou, "Local versus global models for effort-aware defect prediction," in *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 2016, pp. 178–187.
[34] W. Fu and T. Menzies, "Revisiting unsupervised learning for defect prediction," in *FSE*. ACM, 2017, p. to appear.
[35] Q. Huang, X. Xia, and D. Lo, "Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction," in *ICSME*. IEEE, 2017, p. to appear.
[36] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *ICSE*. IEEE Press, 2015, pp. 789–800.
[37] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *TSE*, vol. 38, no. 6, pp. 1276–1304, 2012.
[38] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets," in *ASE*. IEEE, 2015, pp. 452–463.
[39] P. S. Bishnu and V. Bhattacherjee, "Software fault prediction using quad tree-based k-means clustering algorithm," *IEEE Transactions on knowledge and data engineering*, vol. 24, no. 6, pp. 1146–1150, 2012.
[40] B. Fritzke *et al.*, "A growing neural gas network learns topologies," *Advances in neural information processing systems*, vol. 7, pp. 625–632, 1995.
[41] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006, pp. 8–17.
[42] G. Koru, H. Liu, D. Zhang, and K. El Emam, "Testing the theory of relative defect proneness for closed-source software," *Empirical Software Engineering*, vol. 15, no. 6, pp. 577–598, 2010.
[43] A. G. Koru, K. El Emam, D. Zhang, H. Liu, and D. Mathew, "Theory of relative defect proneness," *Empirical Software Engineering*, vol. 13, no. 5, p. 473, 2008.
[44] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, 2010.
[45] T. M. Khoshgoftaar and E. B. Allen, "Ordering fault-prone software modules," *Software Quality Journal*, vol. 11, no. 1, pp. 19–37, 2003.
[46] T. Mende and R. Koschke, "Revisiting the evaluation of defect prediction models," in *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. ACM, 2009, p. 7.
[47] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *TSE*, vol. 34, no. 4, pp. 485–496, 2008.
[48] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *TSE*, vol. 39, no. 6, pp. 757–773, 2013.
[49] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. ACM, 2010, p. 9.
[50] E. G. Jelihovschi, J. C. Faria, and I. B. Allaman, "Scottknott: a package for performing the scott-knott clustering algorithm in r," *TEMA (São Carlos)*, vol. 15, no. 1, pp. 3–17, 2014.
[51] J. Ferreira, A. Zwinderman *et al.*, "On the benjamini–hochberg method," *The Annals of Statistics*, vol. 34, no. 4, pp. 1827–1849, 2006.
[52] J. Romano, J. D. Kromrey, J. Coraggio, J. Skowronek, and L. Devine, "Exploring methods for evaluating group differences on the nsse and other surveys: Are the t-test and cohensd indices the most appropriate choices," in *annual meeting of the Southern Association for Institutional Research*, 2006.