Information and Software Technology 131 (2021) 106452

Contents lists available at ScienceDirect

Information and Software Technology

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

Feature selection and embedding based cross project framework for identifying crashing fault residence

Zhou Xu [a,b], Tao Zhang [c], Jacky Keung [d], Meng Yan [*,a,b], Xiapu Luo [*,e], Xiaohong Zhang [a,b], Ling Xu [a,b], Yutian Tang [f]

[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, Macau University of Science and Technology, Macao, China
[d] Department of Computer Science, City University of Hong Kong, Hong Kong, China
[e] Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China
[f] School of Information and Technology, ShanghaiTech University, Shanghai, China

ARTICLE INFO

Keywords:
Crashing fault
Stack trace
Feature selection
Feature embedding
Cross project framework

ABSTRACT

Context: The automatically produced crash reports are able to analyze the root of fault causing the crash (crashing fault for short) which is a critical activity for software quality assurance.

Objective: Correctly predicting the existence of crashing fault residence in stack traces of crash report can speed up program debugging process and optimize debugging efforts. Existing work focused on the collected label information from bug-fixing logs, and the extracted features of crash instances from stack traces and source code for **I**dentification of **C**rashing **F**ault **R**esidence (**ICFR**) of newly-submitted crashes. This work develops a novel cross project ICFR framework to address the data scarcity problem by using labeled crash data of other project for the ICFR task of the project at hand. This framework removes irrelevant features, reduces distribution differences, and eases the class imbalance issue of cross project data since these factors may negatively impact the ICFR performance.

Method: The proposed framework, called **FSE**, combines **F**eature **S**election and feature **E**mbedding techniques. The FSE framework first uses an information gain ratio based feature ranking method to select a relevant feature subset for cross project data, and then employs a state-of-the-art **W**eighted **B**alanced **D**istribution **A**daptation (**WBDA**) method to map features of cross project data into a common space. WBDA considers both marginal and conditional distributions as well as their weights to reduce data distribution discrepancies. Besides, WBDA balances the class proportion of each project data to alleviate the class imbalance issue.

Results: We conduct experiments on 7 projects to evaluate the performance of our FSE framework. The results show that FSE outperforms 25 methods under comparison.

Conclusion: This work proposes a cross project learning framework for ICFR, which uses feature selection and embedding to remove irrelevant features and reduce distribution differences, respectively. The results illustrate the performance superiority of our FSE framework.

1. Introduction

Software plays a non-substitutive role in the current society. Developing high-quality software without faults has always been the pursuit of all software companies. Due to the increasing scale and complexity of programs and some uncontrollable human mistakes in the software development process, software products may contain faults (or bugs) upon release [1,2]. The faults may lead to software crash. Once a crash is triggered, the system will automatically generate a crash report (usually the stack trace) to record the corresponding status information of the program execution at that time. Fixing the fault causing the crash (crashing fault for short) is a critical task for software quality assurance. To achieve this purpose, developers need to effectively identify the position of the crashing fault in the source code. This process is called crash localization (or fault localization) [3].

Generally, crash localization utilizes the information of the stack

* Corresponding authors.
E-mail addresses: mengy@cqu.edu.cn (M. Yan), csxluo@comp.polyu.edu.hk (X. Luo).

https://doi.org/10.1016/j.infsof.2020.106452
Received 8 January 2020; Received in revised form 20 September 2020; Accepted 9 October 2020
Available online 15 October 2020

trace and the source code to find the root cause of the crash for debugging [4]. The stack trace is a set of frame objects that consists of a runtime exception and a list of the function invocations collected at runtime. If the crashing fault resides inside the stack trace, the developers only need to focus on the source code of the functions recorded in the stack trace. When the crashing fault resides outside the stack trace, the developers have to check the function invocation graphs, spending huge efforts to extensively inspect source code [5]. This will seriously hinder the efficiency demand of the crash localization.

To facilitate the crash localization, Gu et al. [5] proposed an automatic method, called CraTer, to predict whether the crashing fault residence is inside the stack trace or not. We call this task **I**dentification of **C**rashing **F**ault **R**esidence (**ICFR**). According to their definition, if the faulty code exactly matches the record information of one frame in the stack trace, it is deemed that the crashing fault resides inside the stack trace, otherwise, outside the stack trace. They extracted a set of features from the stack trace and the source code to characterize the crashing fault. However, their work only focused on ICFR under the within-project scenario, where the performance highly relies on the precondition that sufficient labeled training crash instances are available. As the collection process of software project data (especially for the label information) is costly and may need a considerable amount of time and workload [6], it is not always feasible to obtain abundant labeled training data. An alternative solution to this dilemma is resorting the advantage of the cross project model which utilizes the labeled data of the external project (*a.k.a.* source project) to serve the task of unlabeled data of the project at hand (*a.k.a.* target project). In this paper, we study the ICFR problem in such scenario by proposing a cross project model (or framework).

The typical usage scenario of our cross project ICFR model is to warn developers about crashing faults that are likely to reside in the stack trace in the absence of labeled crash data. As a result, the predicted results can assist developer with the manual crash localization. For example, suppose Alice is a developer in a large project team and her team is developing a new project (e.g., system A) which has no historical development information to collect the labeled crash data. But they have collected some labeled crash data from an old system B they developed before. One day the system A crashed, and Alice got the crash reports which include the stack trace with t frames.

Without our model, Alice and other related developers have to analyze all frames and the sequence of function calls that are recorded in the stack trace. As a result, they need to review a great deal of lines of codes that are from the functions in the call sequence. This would involve in much human labor and increase the debugging efforts.

With our model, Alice can utilize the labeled crash data from system B to predict whether the fault that crashes system A resides in the stack trace or not. After extracting the crash features from the stack trace and source code of system A, if the identification result shows that the crashing fault locates in the stack trace (i.e., the fault code exactly matches the recorded information in one frame), then Alice and other related developers only need to carefully review the t lines of code in the stack trace. As a result, they discover the root cause of the crash and fix it. This can save human labor and promote the debugging process.

The data distribution discrepancy is the major barrier for cross project models to achieve satisfactory performance. Transfer learning is a commonly-used cross project model which aims to minimize the distribution discrepancy across different domains (one project represents an unique domain). **B**alanced **D**istribution **A**daptation (**BDA**) [7] is a novel transfer learning method to embed the cross project data into a common feature space for reducing the data distribution discrepancies. The advantage of BDA is that it not only considers discrepancies of both marginal and conditional probability distributions, but also allocates different weights to them for effectively adapting to various cross project pairs. This is motivated by the fact that, if the data distributions of two projects are similar, the importance of the conditional probability distribution is dominant, whereas if the data distribution of two projects is dissimilar, the marginal probability distribution has higher importance [7]. Since the studied data are class imbalanced (i.e., there is a large difference between the numbers of crash instances in distinct classes), it will hinder the learning model to achieve promising performance. In this work, we employ the **W**eighted **BDA** (**WBDA**) for feature embedding, which supports adjusting the weight of each class during the process of distribution adaptation [7]. This weighting strategy is helpful for eliminating the negative effect of the class imbalance issue to some extent.

In addition, the project data may consist of a large number of features, which is called the curse of dimensionality. This phenomenon can not only slow down the speed of the model training and occupy more storage space, but also make the trained model complex and over-fitting [8]. Besides, among the initial feature set, there may exist some irrelevant features which will deteriorate the performance of the learning model if all features are used [9]. In this case, feature selection techniques are used to reduce the feature dimension to solve the above problems. More specifically, such techniques select a feature subset, which is important to distinguish instances with different class labels, to replace the original ones. This process can both reduce feature dimensions and retain relevant features. In this work, we use a typical feature selection method which ranks the features according to their importance degrees based on the information **G**ain **R**atio(**GR**) values. GR based feature ranking method is a variant version of **I**nformation **G**ain (IG) based method. These two feature selection methods have been successfully used in previous studies of software engineering domain [10,11], and can achieve comparative performance among the ranking based feature selection family [12].

As the sizes of different features vary even in the same project, normalization techniques are essentially applied to rescale the cross project data into a specific interval before being fed into a machine learning model. This data preprocessing procedure is beneficial to the following learning performance [13]. Previous study showed that the choice of the normalization techniques and the distribution characteristics of different domains can greatly impact the performance of certain learning tasks [14]. To alleviate this issue, Nam et al. [15] proposed a heuristic strategy to adaptively determine an appropriate normalization technique based on the distribution characteristics of the cross project data. However, the impact of normalization techniques on the ICFR performance has not been explored yet. In this work, we make the first attempt to find a better normalization strategy for the crash data, and investigate whether or not the normalization adaption selection in [15] also works well on our cross project ICFR model.

We conduct experiments on a benchmark dataset that consists of crash data from 7 open-source software projects, and employ 6 indicators to evaluate the ICFR performance of our proposed cross project learning framework FSE. The experimental results show that our FSE framework with the most commonly used normalization scheme is sufficient to obtain good performance. By comparing with the best average indicator values among 5 variant methods and 5 downgraded methods, our FSE framework has small performance improvements on all indicators except in terms of the best average AUC value among 5 variant methods. By comparing with the best average indicator values among 15 cross project models from other domains, our FSE framework achieves the average improvements of 41.4%, 28.1%, 27.4%, 119.4%, and 20.4% in terms of F-measure for crash instances inside the stack trace, g-mean, Balance, MCC, and AUC, respectively. But the average F-measure for crash instances outside the stack trace by FSE is lower than 6 methods under comparison.

This paper is an extended version towards our previous study [16] which has been published as a conference paper. There are 5 main differences among the two papers: (1) We address the class imbalance issue in the process of the distribution adaptation which is not considered in the conference version. This operation helps to reduce the negative impact of the difference between the number of crash instances from two categories on the performance of the learning model. (2) Before performing the feature embedding stage, we employ a classic ranking based

feature selection method to reduce the dimensionality of the cross project data, which is not considered in the conference version. This stage reserves the valuable features to mitigate the negative impact of the unimportant features. (3) We add an indicator (i.e., AUC) independent of the classifier threshold for performance evaluation. (4) We add 12 methods for comparison to investigate the effectiveness of our proposed cross project ICFR framework. (5) We make more in-depth analysis towards the impacts of feature dimensions and parameter values on the ICFR performance of the proposed framework (available at our online materials[1]), which are omitted in the conference paper due to the page limit.

In summary, we have the following contributions:

(1) In this work, we propose a novel cross project framework FSE to address the ICFR problem in data scarcity scenario where the labeled data are not always available for the project at hand.
(2) To remove the negative impacts of some useless features on the ICFR performance, our proposed FSE framework first uses a feature selection technique to select the important features by reserving the relevant ones.
(3) To tackle the distribution discrepancy and class imbalance issue of the cross project data, our FSE framework employs a state-of-the-art WBDA based transfer learning method for feature embedding. WBDA reduces the threat of data distribution inconsistency by incorporating both marginal and conditional distributions together with adaptive weights, and alleviates the class imbalance issue by altering the weight of each class during the process of the distribution adaptation.
(4) We comprehensively evaluate the proposed cross project model on 7 open-source projects with 6 indicators. The experimental results on 42 cross project pairs show the superiority of our devised cross project ICFR framework over 25 baseline methods under comparison.

2. Related work

2.1. Stack trace analysis

Once a crash occurs, an exception is thrown out and a crash report will be automatically recorded by the crash reporting system. The main context of the report is the stack trace of the crash which reports the function invocation sequences during execution. The stack trace is useful to reproduce the crash scenario and find the root cause of the crash [17]. A stack trace can be treated as a set of frame objects. The initial frame reports the exception of the crash and each other frame represents a function invocation. The most recent frame is usually called top frame and the last one is usually called bottom frame [5]. The main elements of each frame (except for the initial one) consist of the class name, function name, and the code line number, which denotes the position of the execution point. Other optional elements include the argument information that relates to the function.

Previous studies analyzed stack traces for different tasks, such as crash report clustering, crash reproduction, and crash localization.

2.1.1. Crash report clustering

This kind of task aims to identify the duplicate crash reports which are caused by the same faults. Dang et al. [18] proposed a crash report clustering method, called ReBucket, based on the similarity of the stack trace measured by the position dependent model. Dhaliwal et al. [19] clustered the crash reports by measuring the similarity of stack traces with the Levenshtein distance.

2.1.2. Crash reproduction

Reproducing the crash with well-designed test cases helps to fully understand its root causes during software debugging. Chen et al. [20] proposed the S_{TAR} method which used the collected stack traces by combining a backward symbolic execution approach and a sequence composition technique. Nayrolles et al. [21,22] analyzed the stack traces with static program slices and directed model checking for crash reproduction. Xuan et al. [17] proposed the M_UC_{RASH} method which combined stack traces, source code, and existing test cases to generate mutated test cases. Soltani et al. [23] proposed a post-failure method, called EvoCrash, for automated crash reproducing. This method reduced the search space by using stack traces to guide a genetic algorithm.

2.1.3. Crash localization

Studies on crash localization (or fault localization) are closely related to our work. This kind of task recommends the developers a set of candidate functions based on their suspicious scores by analyzing the stack traces and source code. Wu et al. [24] proposed the CrashLocator method employing 3 static analysis techniques to deduce the failing execution stack traces, and a term weighting method to calculate the suspicious scores for the functions. Wang et al. [25] proposed 3 crash correlation rules to divide the crash types into different groups and a new fault localization method based on the divided groups. Moreno et al. [26] proposed the Lobster method based on the structural and textual similarities. Wong et al. [27] developed a tool, called BRTracer, by analyzing the segmentation of the source code and stack traces for bug reports based fault localization.

Different from the above fault localization studies which returned the potential faulty functions to the developers, Gu et al. [5] proposed the CraTer method to determine whether the location of the faulty code is consistent with the recorded information (including the class name, function name, and code line number) of one frame in the stack trace. From the point of view, the work of Gu et al. can be viewed as a fine-grained identification at the code line level rather than the function level. Since Gu et al. [5] only considered the ICFR task in the same project setting, in this work, we extend their work to cross project setting since the labeled crash data are usually scarce in single project case.

2.2. Cross project learning tasks

Though there are no studies for cross project ICFR yet, researchers have applied cross project models to other software engineering tasks, such as defect prediction, effort estimation, change prediction, and logging prediction.

Cross project defect (or fault) prediction utilizes the fault data of the external project (*a.k.a.* source project) to predict whether the software entities (a function, class, or file) in the target project are fault-prone. Prior studies have proposed different cross project models for this task, such as instance filtering based methods [28–31], transfer learning based methods[14,15,32,33], and classifier combination based methods [34–37]. The studies of cross project fault prediction compose the most active research branch in software engineering field [38,39].

Cross project effort estimation [40,41] estimates the efforts required to develop a target project with the aid of the labeled data from other projects. Cross project change prediction [42–44] determines whether a class in a target project is likely to change in its next release with the help of the labeled data from other projects. Cross project logging prediction [45,46] employs the labeled data from other projects to automatically predict the code constructs that need to be logged in the target project.

Different from the above studies, in this work, we make the first attempt to develop a transfer learning based cross project model for the ICFR task.

2.3. Fuzz testing based crash analysis

The crash instance data in our used benchmark dataset are generated

[1] https://github.com/sepine/IST-2020

by one single seeded fault using mutation testing. The similar work is the fuzz testing based crash analysis study. Fuzz testing is also a software testing technique that uses random data (called fuzz) as program inputs in attempt to make the software system behave unexpectedly and crash for revealing reliability issues [47]. Zhao et al. [48] used black-box fuzz testing on multiple Linux programs to collect data which consisted of 4000 crashes. Shahriar et al. [49] found that fuzz testing can effectively detect the crashes causing by the memory leak in Android applications. Zalewski [50] pointed out that it was difficult to evaluate the exploitability using some crashes generated by fuzz testing without many efforts in debugging and code analysis. Pudas [51] investigated how to improve the performance of uniqueness detection for the crashes in fuzz testing. Chen et al. [52] proposed a method called $\mathrm{H}_{\mathrm{AWKEYE}}$ based on greybox fuzz testing to reveal crashes. The experimental results showed that their method was 7 times faster than an advance method at exposing certain crashes. Liang et al. [53] conducted an empirical study to investigate the effectiveness of fuzz testing on real industry projects from Huawei company. The experimental results showed that they revealed several unreported faults leading to the system crash. Zhang et al. [54] proposed a hybrid method combining dynamic symbolic execution and coverage-based fuzz testing for bugs discovery. The experimental results on real-world programs showed that their method can expose several previously unknown crashes.

3. Method

3.1. Framework overview

Fig. 1 provides an overview diagram of our proposed FSE framework in this study. More specifically, we first employ normalization technique to preprocess the original data of two projects. Then, we employ GR based feature ranking method to select an important feature subset from the source project data since their labels are known in advance, and keep the same feature in the target project data to ensure the data across projects with the same feature set. Next, we utilize the WBDA based transfer learning technique to map the data of the two projects into a common embedding feature space in which the distribution discrepancies of the across project data are reduced. The mapped labeled crash instances in the source project are used to train a classification model to predict the residence labels of the mapped unlabeled crash instances in the target project.

Below, we introduce the details of the used GR based feature selection method and WBDA based feature embedding method, respectively.

3.2. Gain ratio (GR) based feature selection method

For the original source and target project data, we assume that the source project has plenty of labeled crash instances while the target project only has unlabeled crash instances. We define the original source project data as $\mathcal{D}_{\mathrm{S}}^{'}$ with $d_s^{'}$ crash instances and K ($K = 2$ in our work) classes C_k ($k = 1, 2, \ldots, K$) in which $|C_k|$ denotes the number of crash instances belonging to class C_k, that is $\sum_{k=1}^{K}|C_k| = d_s^{'}$. For a given feature A with n different values, the original source project data $\mathcal{D}_{\mathrm{S}}^{'}$ is divided into n subset $\mathcal{D}_{\mathrm{S}_1}^{'}, \mathcal{D}_{\mathrm{S}_2}^{'}, \ldots, \mathcal{D}_{\mathrm{S}_n}^{'}$ according to the values of feature A in which $|\mathcal{D}_{\mathrm{S}_i}^{'}|$ denotes the number of crash instances in $\mathcal{D}_{\mathrm{S}_i}^{'}$, that is $\sum_{i=1}^{n}|\mathcal{D}_{\mathrm{S}_i}^{'}| = d_s^{'}$. In addition, let $\mathcal{D}_{\mathrm{S}_{ik}}^{'}$ denote the set which consists of the crash instances in subset $\mathcal{D}_{\mathrm{S}_i}^{'}$ with class C_k in which $|\mathcal{D}_{\mathrm{S}_i^{'}}|$ denotes the number of crash instances in $\mathcal{D}_{\mathrm{S}_i^{'}}$. In order to calculate the GR value of the given feature A, we need to calculate two entropy values first as follows:

The entropy $H(\mathcal{D}_{\mathrm{S}}^{'})$ of the original source project data $\mathcal{D}_{\mathrm{S}}^{'}$ is defined as

$$\mathrm{H}(\mathcal{D}_{\mathrm{S}}^{'}) = -\sum_{k=1}^{K}\frac{|C_k|}{|\mathcal{D}_{\mathrm{S}}^{'}|}\log_2\frac{|C_k|}{|\mathcal{D}_{\mathrm{S}}^{'}|} \tag{1}$$

Entropy $H(\mathcal{D}_{\mathrm{S}}^{'})$ represents a measure of uncertainty for $\mathcal{D}_{\mathrm{S}}^{'}$.

The conditional entropy $H(\mathcal{D}_{\mathrm{S}}^{'}|A)$ of the original source project data $\mathcal{D}_{\mathrm{S}}^{'}$ given feature A is defined as

$$\mathrm{H}(\mathcal{D}_{\mathrm{S}}^{'}|A) = \sum_{i=1}^{n}\frac{|\mathcal{D}_{\mathrm{S}_i}^{'}|}{|\mathcal{D}_{\mathrm{S}}^{'}|}H\left(\mathcal{D}_{\mathrm{S}_i}^{'}\right) = -\sum_{i=1}^{n}\frac{|\mathcal{D}_{\mathrm{S}_i}^{'}|}{|\mathcal{D}_{\mathrm{S}}^{'}|}\sum_{k=1}^{K}\frac{|\mathcal{D}_{\mathrm{S}_{ik}}^{'}|}{|\mathcal{D}_{\mathrm{S}_i}^{'}|}\log_2\frac{|\mathcal{D}_{\mathrm{S}_{ik}}^{'}|}{|\mathcal{D}_{\mathrm{S}_i}^{'}|} \tag{2}$$

The conditional entropy $H(\mathcal{D}_{\mathrm{S}}^{'}|A)$ represents the uncertainty of $\mathcal{D}_{\mathrm{S}}^{'}$ given the condition that feature A is known.

Then the IG value of feature A is calculated as

$$\mathrm{IG}(\mathcal{D}_{\mathrm{S}}^{'}, A) = \mathrm{H}(\mathcal{D}_{\mathrm{S}}^{'}) - \mathrm{H}(\mathcal{D}_{\mathrm{S}}^{'}|A) \tag{3}$$

The IG value $\mathrm{IG}(\mathcal{D}_{\mathrm{S}}^{'}, A)$ represents the reduction degree of uncertainty for the information of $\mathcal{D}_{\mathrm{S}}^{'}$ when knowing the information of feature A, i.e., the amount of information the feature can provide about whether the crash instance is inside the stack trace (short for InTrance) or outside the stack trace (short for OutTrace). However, the main drawback of IG based feature selection method is that it tends to select the features with a large range of values [11,12]. The alternative solution is to replace IG with GR which overcomes the bias by normalizing the feature's contribution to distinguish the data [11].

Let define $\mathrm{H}_A(\mathcal{D}_{\mathrm{S}}^{'})$ as the entropy of $\mathcal{D}_{\mathrm{S}}^{'}$ in terms of feature A as follows

$$\mathrm{H}_A(\mathcal{D}_{\mathrm{S}}^{'}) = -\sum_{i=1}^{n}\frac{|\mathcal{D}_{\mathrm{S}_i}^{'}|}{|\mathcal{D}_{\mathrm{S}}^{'}|}\log_2\frac{|\mathcal{D}_{\mathrm{S}_i}^{'}|}{|\mathcal{D}_{\mathrm{S}}^{'}|} \tag{4}$$

Then the GR value of feature A is calculated as

$$\mathrm{GR}(\mathcal{D}_{\mathrm{S}}^{'}, A) = \frac{\mathrm{IG}(\mathcal{D}_{\mathrm{S}}^{'}, A)}{\mathrm{H}_A(\mathcal{D}_{\mathrm{S}}^{'})} \tag{5}$$

For each feature in the original source project data, we can calculate its corresponding GR value. GR based feature selection method ranks these methods according to their GR values and selects a specific number or percentage features as the final subset. Let assume that d_s is the number of selected features from the original source project data and define the final simplified source project data as $\mathcal{D}_{\mathrm{S}}$. In order to ensure that the features of the cross project data are one-to-one correspondence,

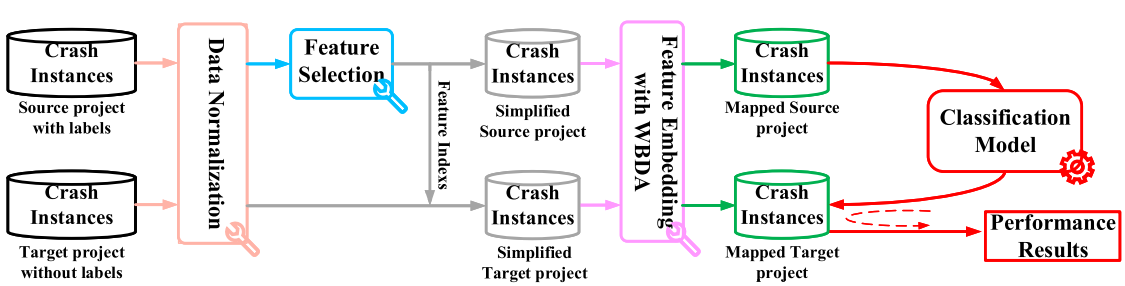

Fig. 1. An overview diagram of our proposed FSE framework.

we also keep the same d_s features for the original target project data and define the final simplified target project data as $\mathcal{D}_{\mathrm{T}}$.

3.3. Balanced distribution adaptation (BDA) model

After the GR based feature selection process in the first step, we obtain the simplified source project and target project data with d_s features as $\mathcal{D}_{\mathrm{S}}$ and $\mathcal{D}_{\mathrm{T}}$, respectively. Specifically, the source project $\mathcal{D}_{\mathrm{S}}$ contains a feature matrix $X_S = x_s^i|_{i=1}^{n_s} \in \mathbb{R}^{n_s \times d_s}$ and a label matrix $Y_S = y_s^i|_{i=1}^{n_s} \in \mathbb{R}^{n_s \times 1}$, where x_s^i represents the ith crash instance in X_S, y_s^i represents the corresponding label, n_s represent the number of crash instances. y_s^i is 'InTrace' if x_s^i exactly matches one of the frame records in the stack trace, otherwise 'OutTrace'. Similarly, the target project $\mathcal{D}_{\mathrm{T}}$ contains a feature matrix $X_T = x_t^i|_{i=1}^{n_t} \in \mathbb{R}^{n_t \times d_s}$, where x_t^i represents the ith crash instance in X_T, n_t represent the number of crash instances, respectively. The corresponding label vector $Y_T = y_t^i|_{i=1}^{n_t}$ is what we are seeking for. In addition, let assume that $\mathcal{X}_s$ ($\mathcal{X}_t$) and $\mathcal{Y}_s$ ($\mathcal{Y}_t$) represent the feature space and label space of the source (target) project, respectively.

In the context of cross project ICFR, $\mathcal{X}_s = \mathcal{X}_t$ and $\mathcal{Y}_s = \mathcal{Y}_t$, which means that the two projects have the same feature space and label space respectively, but $\mathcal{P}(x_s) \neq \mathcal{P}(x_t)$ and $\mathcal{P}(y_s|x_s) \neq \mathcal{P}(y_t|x_t)$, which means that the two projects have different marginal distribution and conditional distribution, respectively. The goal of BDA [7] is to learn a common feature space in which the two distribution discrepancies, i.e., $d(\mathcal{P}(x_s), \mathcal{P}(x_t))$ and $d(\mathcal{P}(y_s|x_s), \mathcal{P}(y_t|x_t))$, are minimal.

The simplest way to reduce the difference between $\mathcal{D}_S$ and $\mathcal{D}_T$ is to optimize the two distribution discrepancies with the same weight as follows:

$$d(\mathcal{D}_S, \mathcal{D}_T) = d(\mathcal{P}(x_s), \mathcal{P}(x_t)) + d(\mathcal{P}(y_s|x_s), \mathcal{P}(y_t|x_t)). \tag{6}$$

For the data of the two projects, the margin distribution is more important when they are dissimilar, otherwise the conditional distribution should be emphasized [7]. Thus just combining the two terms with the same weight in Eq. (6) could not well adapt to all cross project data. BDA alleviates this issue by assigning adaptive weights to the two terms for different cross project pairs. It is formulated as follows:

$$d(\mathcal{D}_S, \mathcal{D}_T) = (1-\mu) d(\mathcal{P}(x_s), \mathcal{P}(x_t)) + \mu d(\mathcal{P}(y_s|x_s), \mathcal{P}(y_t|x_t)), \tag{7}$$

where $\mu \in [0, 1]$ measures the importance of the two terms. $\mu > 0.5$ ($\mu < 0.5$) means that the conditional (marginal) distribution is more important.

However, the label set of the target project Y_T is unknown beforehand, thus the term $\mathcal{P}(y_t|x_t)$ could not be calculated. Long et al. [55] suggested to use class conditional distribution $\mathcal{P}(x_t|y_t)$ to replace conditional distribution as long as that data samples are sufficient. The alternative term can be calculated by training a classifier on $\mathcal{D}_S$ and predicting on $\mathcal{D}_T$. It is worth noting that, since the initial outputs may be not reliable, these output labels are iteratively refined until they are stabilized.

By using the **M**aximum **M**ean **D**iscrepancy (**MMD**) method [56] to calculate the two terms, i.e., $d(\mathcal{P}(x_s), \mathcal{P}(x_t))$ and $d(\mathcal{P}(x_s|y_s), \mathcal{P}(x_t|y_t))$, Eq. (7) is rewritten as

$$\begin{aligned} d(\mathcal{D}_S, \mathcal{D}_T) &= (1-\mu) \| \frac{1}{n_s} \sum_{i=1}^{n_s} x_s^i - \frac{1}{n_t} \sum_{j=1}^{n_t} x_t^j \|_{\mathcal{H}}^2 \\ &+ \mu \sum_{c=1}^{C} \| \frac{1}{n_s^c} \sum_{x_s^i \in \mathcal{D}_S^{(c)}} x_s^i - \frac{1}{n_t^c} \sum_{x_t^j \in \mathcal{D}_T^{(c)}} x_t^j \|_{\mathcal{H}}^2, \end{aligned} \tag{8}$$

where $\mathcal{H}$ represents the reproducing kernel Hilbert space, C represent the number of distinct labels, $\mathcal{D}_S^{(c)}$ ($\mathcal{D}_T^{(c)}$) represents the crash instances with label c in source (target) project, n_s^c (n_t^c) represents the number of crash instances in $\mathcal{D}_S^{(c)}$ ($\mathcal{D}_T^{(c)}$).

By using the matrix tricks and regularization, Eq. (8) is converted to

$$\begin{aligned} &\min_A \operatorname{tr}\left(A^\top X \left((1-\mu) M_0 + \mu \sum_{c=1}^{C} M_c \right) X^\top A \right) + \lambda \| A \|_F^2 \\ &\text{s.t. } A^\top X H X^\top A = I, 0 \leq \mu \leq 1, \end{aligned} \tag{9}$$

where the first term adapts the weights of the two distributions, and the second term is a regularization term. The two constraint terms are used to maintain the inner structure properties of the original data for the transformed one $A^\top X$ and control the μ value, respectively. In addition, X represents the input feature matrix combining X_S and X_T, A represents a transformation matrix, I represents identity matrix with size $(n_s + n_t) \times (n_s + n_t)$, $H = I - (1/n)\boldsymbol{I}$ represents a centering matrix, and $\| A \|_F^2$ represents the Frobenius norm of A. M_0 and M_c represent MMD matrices as follows:

$$(M_0)_{ij} = \begin{cases} \frac{1}{n_s^2}, & x_i, x_j \in \mathcal{D}_S \\ \frac{1}{n_t^2}, & x_i, x_j \in \mathcal{D}_T \\ -\frac{1}{n_s n_t}, & \text{otherwise}, \end{cases} \tag{10}$$

$$(M_c)_{ij} = \begin{cases} \frac{1}{n_s^{c2}}, & x_i, x_j \in \mathcal{D}_S^{(c)} \\ \frac{1}{n_t^{c2}}, & x_i, x_j \in \mathcal{D}_T^{(c)} \\ -\frac{1}{n_s^c n_t^c}, & \begin{cases} x_i \in \mathcal{D}_S^{(c)}, x_j \in \mathcal{D}_T^{(c)} \\ x_i \in \mathcal{D}_T^{(c)}, x_j \in \mathcal{D}_S^{(c)} \end{cases} \\ 0, & \text{otherwise}. \end{cases} \tag{11}$$

By using the Lagrange multiplier method [57], the Lagrange function of Eq. (9) is

$$\begin{aligned} L = \ & \operatorname{tr}\left(A^\top X \left((1-\mu) M_0 + \mu \sum_{c=1}^{C} M_c \right) X^\top A \right) \\ & + \lambda \| A \|_F^2 + \operatorname{tr}((I - A^\top X H X^\top A)\Phi), \end{aligned} \tag{12}$$

where $\Phi = (\phi_1, \ldots, \phi_d)$ represents the Lagrange multiplier. By setting the first-order derivative of L towards A to 0, Eq. (12) is converted to a generalized eigen-decomposition problem as

$$\left(X \left((1-\mu) M_0 + \mu \sum_{c=1}^{C} M_c \right) X^\top + \lambda I \right) A = X H X^\top A \Phi. \tag{13}$$

The result of Eq. (13) is the transformation matrix A, which is used to convert the original data of the two projects.

We provide an example of simulated data to illustrate the feature transformation effect of the BDA method. For the source project, we generate 130 crash instances outside the stack trace (red circles) from a mixture of Gaussian with means (2, 3.5), and 50 crash instances inside the stack trace (blue circles) from a mixture of Gaussian with means (6.5, 2.5), as showed in Fig. 2(a). To reflect the distribution differences across projects, for the target project, we generate 120 crash instances outside the stack trace (red pentagrams) from a mixture of Gaussian with means (4, 5.5), and 60 crash instances inside the stack trace (blue pentagrams) from a mixture of Gaussian with means (6, 1), as showed in Fig. 2(b). Fig. 2(c) depicts the mapped data of two projects by the BDA method with the equal weight in the common feature space. From Fig. 2, we observe that the new data of the two projects mainly locate in two regions marked with black rectangles in the embedding feature space, which reduces the data distribution discrepancies between the two projects.

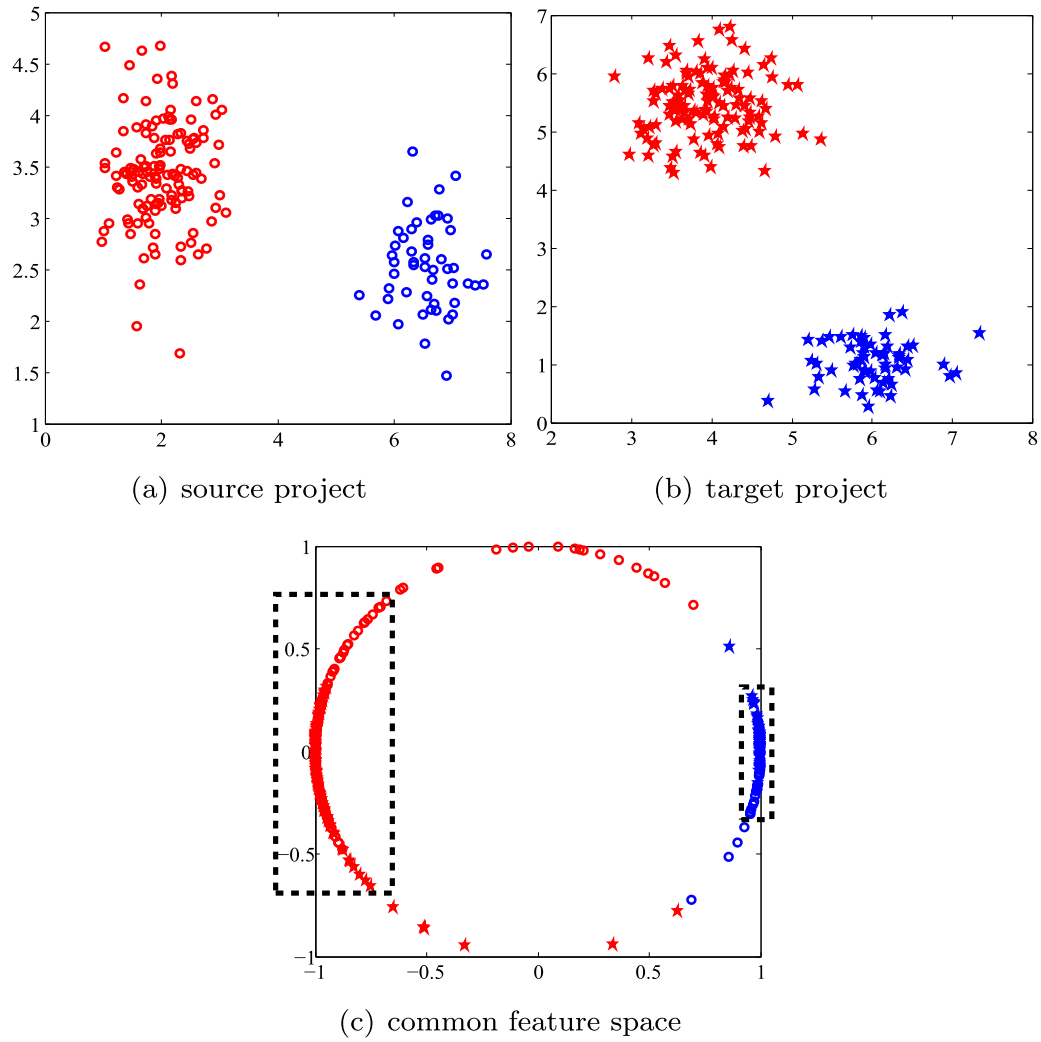

Fig. 2. An example of the feature transformation effect by BDA.

3.4. Weighted BDA (WBDA) based feature embedding method

Although the BDA method considers the different importance degrees of the marginal and conditional distribution discrepancies to reduce the distribution inconsistence across the project data, it supposes that the class probability in each project data is similar when using the class conditional distribution $\mathcal{P}(x_t|y_t)$ to replace the conditional distribution $\mathcal{P}(y_t|x_t)$. It means that the BDA method does not take the class imbalance issue into consideration. To address this issue, in this work, we employ the weighted version of BDA, i.e., WBDA [7], to perform the feature transformation by utilizing a more accuracy approximation to calculate the conditional distribution difference as follows:

$$\begin{aligned} d(\mathcal{P}(x_s|y_s),\mathcal{P}(x_t|y_t)) &= ||\frac{P(y_s)}{P(x_s)}P(x_s|y_s)-\frac{P(y_t)}{P(x_t)}P(x_t|y_t)||^2_{\mathcal{H}} \\ &= ||\alpha_s P(x_s|y_s)-\alpha_t P(x_t|y_t)||^2_{\mathcal{H}} \end{aligned} \tag{14}$$

where α_s and α_t are approximated to the class prior of the source and target project data, respectively. From this point of view, WBDA considers the class proportion differences of the cross project data.

To calculate this conditional difference, a weight matrix W_c for each class is constructed as follows:

$$(W_c)_{ij} = \begin{cases} \frac{P(y_s^{(c)})}{n_s^{c2}}, & x_i,x_j \in \mathcal{D}_S^{(c)} \\ \frac{P(y_t^{(c)})}{n_t^{c2}}, & x_i,x_j \in \mathcal{D}_T^{(c)} \\ -\frac{\sqrt{P(y_s^{(c)})P(y_t^{(c)})}}{n_s^c n_t^c}, & \begin{cases} x_i \in \mathcal{D}_S^{(c)}, x_j \in \mathcal{D}_T^{(c)} \\ x_i \in \mathcal{D}_T^{(c)}, x_j \in \mathcal{D}_S^{(c)} \end{cases} \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

where $P(y_s^{(c)})$ and $P(y_t^{(c)})$ are the class prior on class c in the source and target project data, respectively.

Then, using the matrix tricks and regularization with Eqs. (15), (9) is converted to the optimization problem of WBDA as follows:

$$\begin{aligned} &\min_A \operatorname{tr}\left(A^\top X\left((1-\mu)M_0+\mu\sum_{c=1}^{C}W_c\right)X^\top A\right)+\lambda\|A\|_F^2 \\ &\text{s.t. } A^\top XHX^\top A = I, 0\le\mu\le 1, \end{aligned} \tag{16}$$

The solving process of the optimal transformation matrix A is the same as the one of the BDA method as described in Section 3.3. The only difference between Eq. (9) for BDA and Eq. (16) for WBDA is that the track M_c is replaced by W_c. In other words, in addition to considering the number of crash instances in each class as M_c does, W_c also takes the

class prior of each class into account.

4. Experimental setup

4.1. Benchmark dataset

In this work, we employ a publicly available benchmark dataset provided by Gu et al. [5] to evaluate the performance of our proposed cross project FSE framework for ICFR task. The benchmark dataset is collected from 7 open source Java projects: Apache Commons **Codec**, Apache Commons **Collections**, Apache Commons **IO, Jsoup, JSqlParser, Mango**, and **Ormlite-Core**. Table 1 describes the basic statistics of the 7 projects, including the version number, the total number of generated mutants (# Mutants), the number of remained crashes after removing useless mutants (# Crashes), the number of crash instances inside (# InTrace) and outside (# OutTrace) the stack trace, and the ratio of $\frac{\#\text{OutTrace}}{\#\text{InTrace}}$. The data collection consists of 3 main steps: crash generation, feature (independent variable) extraction, and crashing fault residence (dependent variable) labeling. We briefly describe each step as follows:

4.1.1. Crash generation

(1) Fault Generation via Program Mutation

Since it is non-trivial to reproduce the real-world crashes, Gu et al. [5] simulated the crashes by seeding faults into the real-world projects. More specifically, they applied a state-of-the-art mutation testing tool, the PIT system[2], to generate single-point mutations. In other words, a small change is made to the source code to form a program mutant in each round. These mutations are derived from 7 default mutation operators[3] in the PIT system, including conditionals boundary mutator, increments mutator, invert negatives mutator, math mutator, negate conditionals mutator, return values mutator, and void method call mutator.

(2) Removing Mutants Causing No Crash

After obtaining these program mutants, 4 rules were employed to remove some program mutants which do not crash the program. The rules include removing the mutants that pass all test cases, the mutants whose stack traces only contain *AssertionFailedError, ComparisonFailure*, or test cases.

4.1.2. Independent variable extraction

In order to characterize each crash instance (i.e., crashing fault), Gu et al. [5] extracted 89 features from the stack traces and source code as the independent variables in tota. The brief descriptions of these features are available at our online materials.

4.1.3. Dependent variable labeling

The 3 main terms recorded in the frame (including class name, function name and line number) are used to label each crash instance. More specifically, if the information of the faulty code exactly matches the 3 terms in one frame, then the residence of the crash instance is deemed as inside the stack trace and labeled as 'InTrace', otherwise labeled as 'OutTrace'. The process of label collection is done automatically by checking the bug-fixing logs which are generated when the original authors in [5] wrote programs to produce the mutants.

Table 1
Statistic information of the 7 projects.

Project	Version	# Mutants	# Crashes	# InTrace	# OutTrace	Ratio
Codec	1.1	2901	610	177	433	2.45
Collections	4.1	6650	1350	273	1077	3.95
IO	2.5	3337	686	149	537	3.60
Jsoup	1.11.1	2657	601	120	481	4.01
JSqlParser	0.9.7	8757	647	61	586	9.61
Mango	1.5.4	5149	733	53	680	12.83
*Ormlite-Core	5.1	3563	1303	326	977	3.00

[2] http://pitest.org/
[3] http://pitest.org/quickstart/mutators/#INCREMENTS

4.2. Performance indicator

As the goal of ICFR is to determine the label of a crash instance as 'InTrace' or 'OutTrace', it is a typical binary classification problem. Note that, in this work, we cannot simply say which type of crash instance is the positive one, we calculate indicators towards the two kinds of instances. There are 4 prediction outputs for the ICFR task:

- a crash with labeled t (t is 'InTrace' or 'OutTrace') is predicted as t,
- a crash with labeled t is predicted as t′ (t′ is the opposite of t),
- a crash with labeled t′ is predicted as t′,
- a crash with labeled t′ is predicted as t

The numbers of crash instances that with the above 4 outputs are called **T**rue **P**ositive (**TP**(t)), **F**alse **N**egative (**FN**(t)), **T**rue **N**egative (**TN**(t)), and **F**alse **P**ositive (**FP**(t)), respectively. First, we give the definitions of 3 basic terms, i.e., Precision, **P**robability of **D**etection (**PD** or Recall), and **P**robability of **F**alse alarm (**PF**) as follows:

Precision measures the ratio of crashes with label *t* that are correctly predicted to the total number of crashed that are predicted as *t*, i.e., $\text{Precision}(t) = \frac{\text{TP}(t)}{\text{TP}(t)+\text{FP}(t)}$. PD or Recall measures the ratio of the crashes with label *t* that are correctly predicted to the total number of crashes with label *t*, i.e., $\text{PD}(t) = \text{Recall}(t) = \frac{\text{TP}(t)}{\text{TP}(t)+\text{FN}(t)}$. PF measures the ratio of the crash with label *t* that are incorrectly predicted to the total number of crash with label *t*, i.e., $\text{PF}(t) = \frac{\text{FP}(t)}{\text{FP}(t)+\text{TN}(t)}$.

F-measure (F) is the harmonic mean of Precision and Recall as

$$\text{F}(t) = \frac{2 \times \text{Precision}(t) \times \text{Recall}(t)}{\text{Precision}(t) + \text{Recall}(t)} \tag{17}$$

g-mean is the geometric mean of PD and (1-PF) as

$$\text{g-mean}(t) = \sqrt{\text{PD}(t) \times (1 - \text{PF}(t))} \tag{18}$$

Balance is a trade-off between Recall and PF as

$$\text{Balance}(t) = 1 - \sqrt{\frac{(0 - \text{PF}(t))^2 + (1 - \text{Recall}(t))^2}{2}} \tag{19}$$

MCC (Matthews correlation coefficient) is a special case of Pearson correlation coefficient considering TP, TN, FP, and FN, which is defined as

$$\text{MCC}(t) = \frac{\text{TP}(t) \times \text{TN}(t) - \text{FP}(t) \times \text{FN}(t)}{\sqrt{(\text{TP}(t)+\text{FP}(t))(\text{TP}(t)+\text{FN}(t))(\text{TN}(t)+\text{FP}(t))(\text{TN}(t)+\text{FN}(t))}} \tag{20}$$

AUC is the **A**rea **U**nder the ROC **C**urve whose horizontal axis represents PF and vertical axis represents PD. The calculation of this indicator is independent of the specific classifier threshold.

In this study, we employ F, g-mean, Balance, MCC and AUC as performance indicators. In ICFR scenario, for each indicator, we need to calculate two performance values which correspond to the crash with different labels ('InTrace' and 'OutTrace'). It is worth noting that, except for F, the other 4 indicator values on two labels are the same. This means that the 4 indicators can be used to comprehensively quantify the overall ability of the method to discriminate the two labels. Thus, we total have 6 indicators in which the two kinds of the F indicator are called F (InTrace) and F(OutTrace).

4.3. Statistic test

To statistically analyze the significant differences among our FSE framework and the baseline methods, we employ Friedman test (significant level at 95%) with an improved Nemenyi [38] to divide the methods into completely nonoverlapping groups. The Friedman test is a non-parametric test for testing the differences between several related samples. In our work, there exists the one-to-one correspondence between each set of results of multiple methods since they are the results on the same cross project pair. Thus, the multiple sets of experimental results of the analyzed methods are related. The improved Nemenyi test is employed in previous studies [58,59] of the software engineering domain.

4.4. Classification model

After obtaining the mapped data of the two projects with the optimal transformation matrix, we need to train a classifier with the mapped labeled source project data. As the aim of this work is not to focus on the impacts of different classifiers on the cross project ICFR performance, thus, in this work, we just use logistic regression as the basic classification model. It is a classic and simple classifier which solves the relationship between the features and labels of the crash instances, and is proved to be effective in other software engineering task [60–62].

4.5. Parameter settings

In the feature selection phase of the FSE framework, we need to select a certain number of relevant features from the data of source and target projects. Without loss of generality, in this work, we just remain half of features with higher GR values. For WBDA method in the feature embedding phase of the FSE framework, the factor μ in Eq. (16) is a project-specific parameter and relates to the similarity degree of the data distributions across projects. However, no effective methods are available to specify it on distinct cross project pairs [7]. In this work, we set 11 μ values, from 0 to 1 with a step of 0.1 and search its optimum option. For the regularization parameter λ in Eq. (16), we set it as 0.1 without any guidance of experience. In addition, after the feature embedding phase, we need to set the reserved feature dimension to obtain the final mapped data which are the inputs of the classification model. In this work, we set the dimension as 15% of the original feature number as suggested in previous studies [10,11].

The experimental scripts, the benchmark dataset, and the discussion of the impacts of different parameter λ values and feature dimension on the performance of proposed FSE framework are available in our online materials.

5. Performance evaluation

5.1. RQ1: How do different data normalization strategies impact the performance of our proposed FSE framework?

Motivation: Nam et al. [15] have stated that different data normalization methods can impact the performance of the cross project model for the defect prediction task. They proposed an adaption selection strategy to select the appropriate normalization technique to transform the data before performing the cross project model. The behind rationale is based on the similarity of the data characteristics (such as the mean, median, min and max values) between the source and target project data. This question is designed to investigate whether or not our cross project ICFR framework is sensitive to different data normalization methods and to find the most suitable one for our data preprocessing.

Method: We employ total 6 normalization techniques in [15] to answer this question. These techniques are derived from 2 widely-used data normalization techniques, i.e., min-max normalization and the z-score normalization. For each feature vector $\boldsymbol{x} = \{x_1, x_2, \ldots, x_m\}$ in the given data, in terms of min-max normalization, $\overline{x_i} = \frac{x_i - \min(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})}$, where min $(\boldsymbol{x})$ and max $(\boldsymbol{x})$ represent the minimum and maximum values of the feature vector $\boldsymbol{x}$ respectively, x_i and $\overline{x_i}$ are the original and normalized ith valued of the feature vector $\boldsymbol{x}$ respectively. In terms of z-score normalization, $\overline{x_i} = \frac{x_i - mean(\boldsymbol{x})}{std(\boldsymbol{x})}$, where $mean(\boldsymbol{x})$ and $std(\boldsymbol{x})$ represent the mean value and standard deviation of the feature vector $\boldsymbol{x}$, respectively. First, we describe the 6 normalization techniques as follows:

N0: Do not use any normalization on the original data.

N1: Applying min-max normalization to each project.

N2: Applying z-score normalization to each project.

N3: Applying z-score normalization to each project with the mean value and standard deviation from the source project.

N4: Applying z-score normalization to each project with the mean value and standard deviation from the target project.

NAS: Normalization **A**daption **S**election utilizes a heuristic strategy to select the optimum normalization option from the above 5 techniques. This heuristic strategy is based on the elements of a data characteristic vector which measures the similarity of the data characteristic between two projects.

Results: Fig. 3 shows the radar chart of the 6 average indicator values across all cross project pairs for our proposed FSE framework with 6 data normalization schemes. In the radar chart, each axis represents an indicator and the location of the point on the axis denotes the average value of the indicator. The further away the point is from the center, the greater the average indicator value is. The 6 average indicator values (i.e., 6 points) of a method are connected into a polygon marking with a specific color. The detailed experimental results are also available in our online materials. From Fig. 3, we observe that our proposed FSE framework with normalization schemes N0 and N1 achieves the worst average performance in terms of F(InTrace), g-mean, Balance, MCC and AUC, but gets the best average F(OutTrace). It means that FSE framework with these two normalization schemes identifies the residences of nearly all the crash instances as label 'OutTrace'. Thus, they make no sense for ICFR task even although they obtain the best average F(OutTrace). In addition, FSE framework with N2, N3, N4 and NAS schemes achieve nearly the same average performance in terms of all indicators.

Fig. 4 visualizes the statistic test results of Friedman test with Nemenyi post-hoc test for our proposed FSE framework with the 6 schemes in terms of the 6 indicators. As the p-values of Friedman test are all less than 0.05, it indicates that performance differences among the 6 schemes can be explained by statistical significance in terms of all 6 indicators. The CD diagrams of Nemenyi test illustrate that our FSE framework with N2, N4, and NAS schemes belong to the top ranking group on 5 indicators except in terms of F(OutTrace). Here, the top ranking group means that the methods in this group have statistically

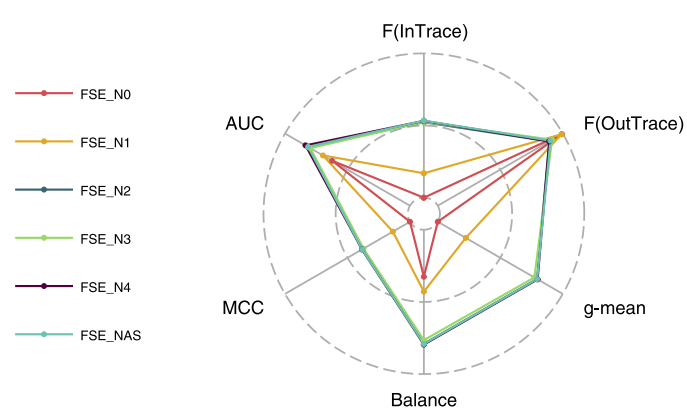

Fig. 3. The radar chart of average indicator values across all cross project pairs for FSE framework with different normalization schemes.

significant differences compared with those in other groups.

Analysis: Directly using the original data set (i.e., N0 option) as the input of the FSE framework results in the worst performance which demonstrates that data normalization is a necessary preprocessing step. The experimental results show that the performance based on z-score normalization is significantly better than that based on min-max normalization, whereas different schemes based on z-score normalization have no significant differences. Overall, normalization schemes N2, N4, and NAS are the better options. This conclusion is somewhat inconsistent with that in [15] which suggested that normalization scheme NAS is the most suitable choice. The reasons are that our FSE method achieves the similar average results on crash data preprocessed by N2 and N4 schemes, and NAS selects the N2 and N4 normalization schemes to preprocess the crash data on many cross project pairs. Thus, there is not much difference between the experimental results on crash data preprocessed by NAS, N2, and N4 schemes. While in [15], normalization schemes selected by NAS may vary on different cross project pairs, which leads to that NAS becomes the better choice to preprocess the data.

Answer: Considering that (1) applying z-score technique to the cross project data individually (i.e., the N2 option) is sufficient to achieve better ICFR performance; (2) it is the most commonly used data preprocessing scheme in previous studies [63,64] without utilizing intricate rules to select the optimal normalization scheme (i.e., the NAS option) for specific cross project pair, in this work, we normalize the cross project data with the N2 option before conducting the FSE framework in the following experiments.

5.2. RQ2:Are the feature selection process and the weighting scheme in the FSE framework helpful to improve the cross project ICFR performance?

Motivation: As FSE is a combined framework of a GR based feature selection process and a weighted feature embedding method, this question is designed to explore whether the feature selection process and the weighting scheme are beneficial for the performance improvement of the proposed FSE framework.

Methods: To answer this question, we employ the methods with and without the feature selection process, and the methods with and without the weighting scheme for comparison. In this work, we call these methods as variant methods of the FSE framework. More specifically, we choose a total of 5 variant methods as follows:

NONE method does not involve any GR based feature selection and transfer learning based feature embedding.

GR method only uses the GR based feature selection without transfer learning based feature embedding.

BDA method only uses the BDA based feature embedding without GR based feature selection.

WBDA method only uses the WBDA based feature embedding without GR based feature selection.

GRBDA method uses the GR based feature selection with BDA based feature embedding.

NONE is the most basic method to investigate the effectiveness of both feature selection and feature embedding. By comparing FSE against GR, we can investigate the effectiveness of feature embedding. By comparing FSE against WBDA and GRBDA against BDA, we can investigate the effectiveness of feature selection. By comparing FSE against GRBDA and WBDA against BDA, we can investigate the effectiveness of the weighting scheme in the feature embedding.

Results: Fig. 5 shows the radar chart of the 6 average indicator values for our proposed FSE framework and 5 variant methods. We draw radar charts for the cross project pairs on each project and across all projects. Thus, we obtain a total of 8 radar charts in which the last one reports the average indicator values across all 42 cross project pairs. More specifically, since we conduct one-to-one cross project experiment, we have 6 sets of experimental results for each performance indicator when one project is selected as the target project. Thus, Fig. 5(a)–(g) present the radar charts of the 6 sets of experimental results for each project which is selected as the target project. In addition, as we conduct experiments on a total of 42 cross project pairs from the 7 projects, Fig. 5 (h) presents the radar charts of the 6 indicators across the 42 pairs. From the figure, we have the following observation:

First, from Fig. 5(h), we find that our FSE framework achieves the best average values on 5 indicators (except in terms of AUC) across 42 cross project pairs compared with the 5 variant methods. The 5 average

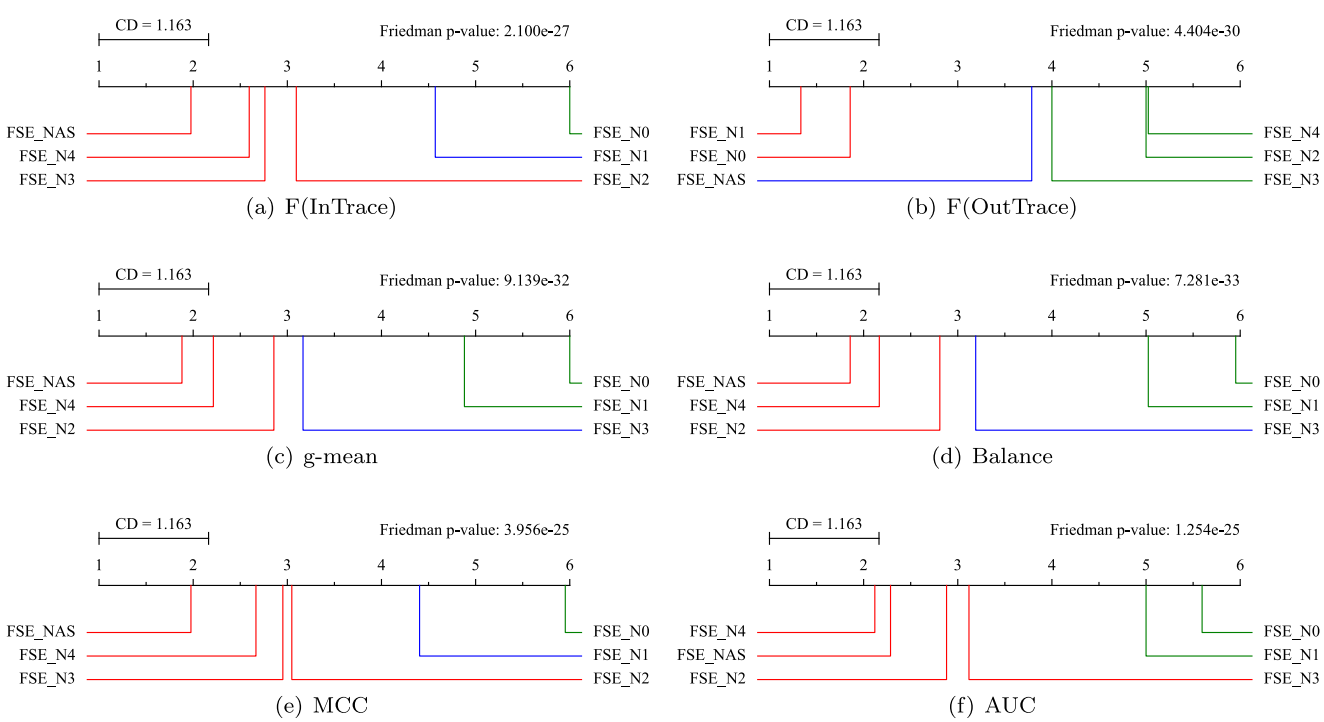

Fig. 4. Statistic test results for FSE framework with 6 normalization schemes in terms of 6 indicators.

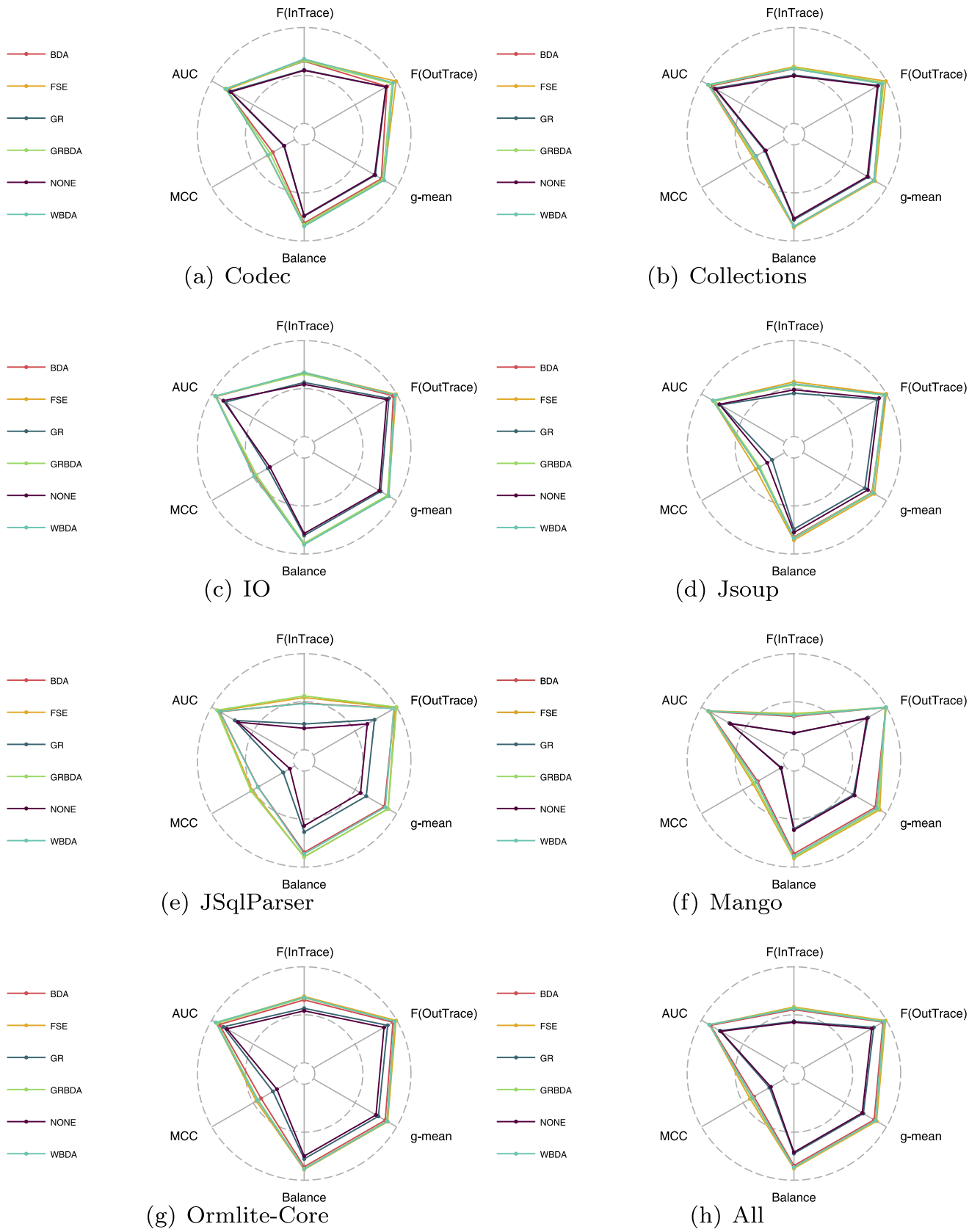

Fig. 5. The radar chart of average indicator values for FSE framework and 5 variant methods on each project and all projects.

indicator values by our FSE framework are much higher than that by NONE and GR, but a little higher than that by WBDA and GRBDA methods. More specifically, compared with the best average indicator values among the 5 variant methods, our FSE framework achieves the average improvements of 1.5%, 1.3%, 1.3%, 1.3%, and 5% in terms of F (InTrace), F(OutTrace), g-mean, Balance, and MCC, respectively. In addition, the average AUC value by FSE is much higher than that by NONE and GR methods, similar to that by BDA and GRBDA methods, and only slightly lower than that by the WBDA method.

Second, from Fig. 5(b), (d), (f), and (g), the results show that average F(InTrace) and Balance values by our FSE framework are better than that by the 5 variant methods on project Collections, Jsoup, Mango, and Ormlite-Core, respectively; from Fig. 5(a)–(d), and (g), the results show that average F(OutTrace) value by our FSE framework is better than that by the 5 variant methods on project Codec, Collections, IO, Jsoup, and Ormlite-Core, respectively; from Fig. 5(b), (d), (f), and (g), the results show that average g-mean value by our FSE framework is better than that by the 5 variant methods on project Collections, Jsoup, Mango, and Ormlite, respectively; from Fig. 5(a), (b), (d), (f), and (g), the results show that average MCC value by our FSE framework is better than that by the 5 variant methods on project Codesc, Collections, Jsoup, Mango, and Ormlite, respectively.

Third, Fig. 6 visualizes the corresponding statistic test results for the 6 methods in terms of the 6 indicators. The p-values of Friedman test (all less than 0.05) indicates that the performance differences among the 6 methods are statistically significant in terms of all indicators. The CD diagrams show that our FSE framework belongs to the top ranking group in terms of all indicators and achieves the best average ranking on 5 indicators except in terms of AUC. But our FSE framework has no significant differences compared with 2 variant methods in terms of F (OutTrace) and 3 variant methods in terms of other 5 indicators.

Analysis: Overall, the performance values of our FSE framework and the WBDA method are better than that of GRBDA and BDA methods, respectively. This indicates that the weighting scheme in the feature embedding is useful for performance improvement. From the point of the average ranking in the CD diagram, our FSE framework, GRBDA, and GR methods perform better than WBDA, BDA, and NONE methods, respectively. This indicates that the GR based feature selection process can promote the ICFR performance. The obvious performance superiority of our FSE frameowrk compared with the GR method manifests the importance of the feature embedding process. In addition, although FSE, WBDA, and GRBDA belong to the same top rank group in the 6 indicators, the specific ranking values of the 3 methods show that our FSE method ranks the first in terms of 5 performance indicators except in terms of AUC. In addition, the detailed results in our online materials show that our FSE method obtains the best average performance across the 42 cross project pairs of the 7 projects in terms of 5 indicators except in terms of AUC and obtains nearly the same average AUC value as WBDA and GRBDA. Overall, FSE is still superior to the two baseline methods, although the performance improvement is small and not significant.

Answer: Both the feature selection process and weighting scheme in the feature embedding can promote the performance of the FSE framework for cross project ICFR task.

5.3. RQ3:Does FSE perform better than the downgraded methods in terms of WBDA?

Motivation: As mentioned in Section 3.4, the WBDA method in our proposed FSE framework incorporates both the marginal and conditional distributions during the feature embedding process. Besides, it assigns different weights to the two distributions for distinct cross project pairs and considers the class imbalanced issue when calculating the conditional distribution discrepancy. This question is designed to study whether our FSE framework is superior to GR combining the downgraded methods of the using transfer learning method WBDA. These methods are the downgraded versions in terms of WBDA, such as the ones that only consider one of the distribution distribution with and without the weighting scheme.

Method: To answer this question, we employ a total of 5 downgraded methods for comparison as follows:

GRTCA method combines GR based feature selection with the feature embedding method **TCA** (**T**ransfer **C**omponent **A**nalysis) [56] that only focuses on narrowing margin distribution discrepancy (i.e., $\mu = 0$ in Eq. (7)).

GRCDT method combines GR based feature selection with the

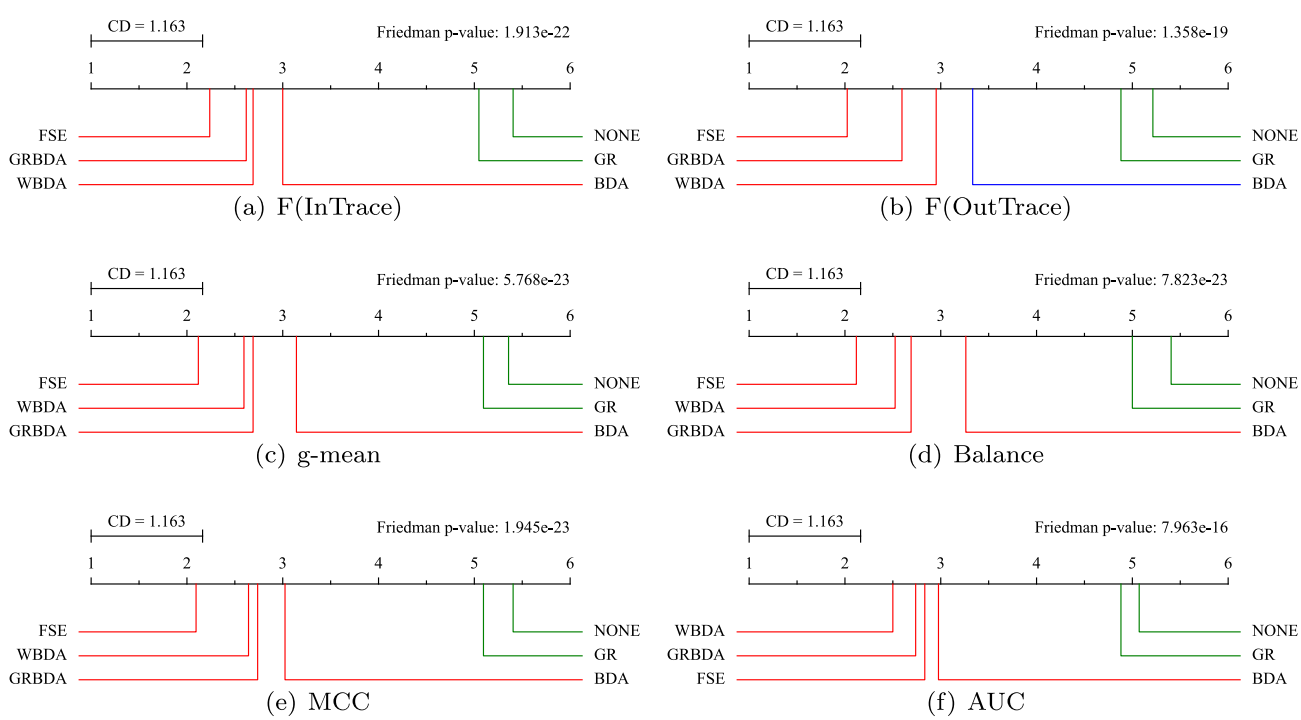

Fig. 6. Statistic test results for FSE framework and 5 variant methods in terms of 6 indicators.

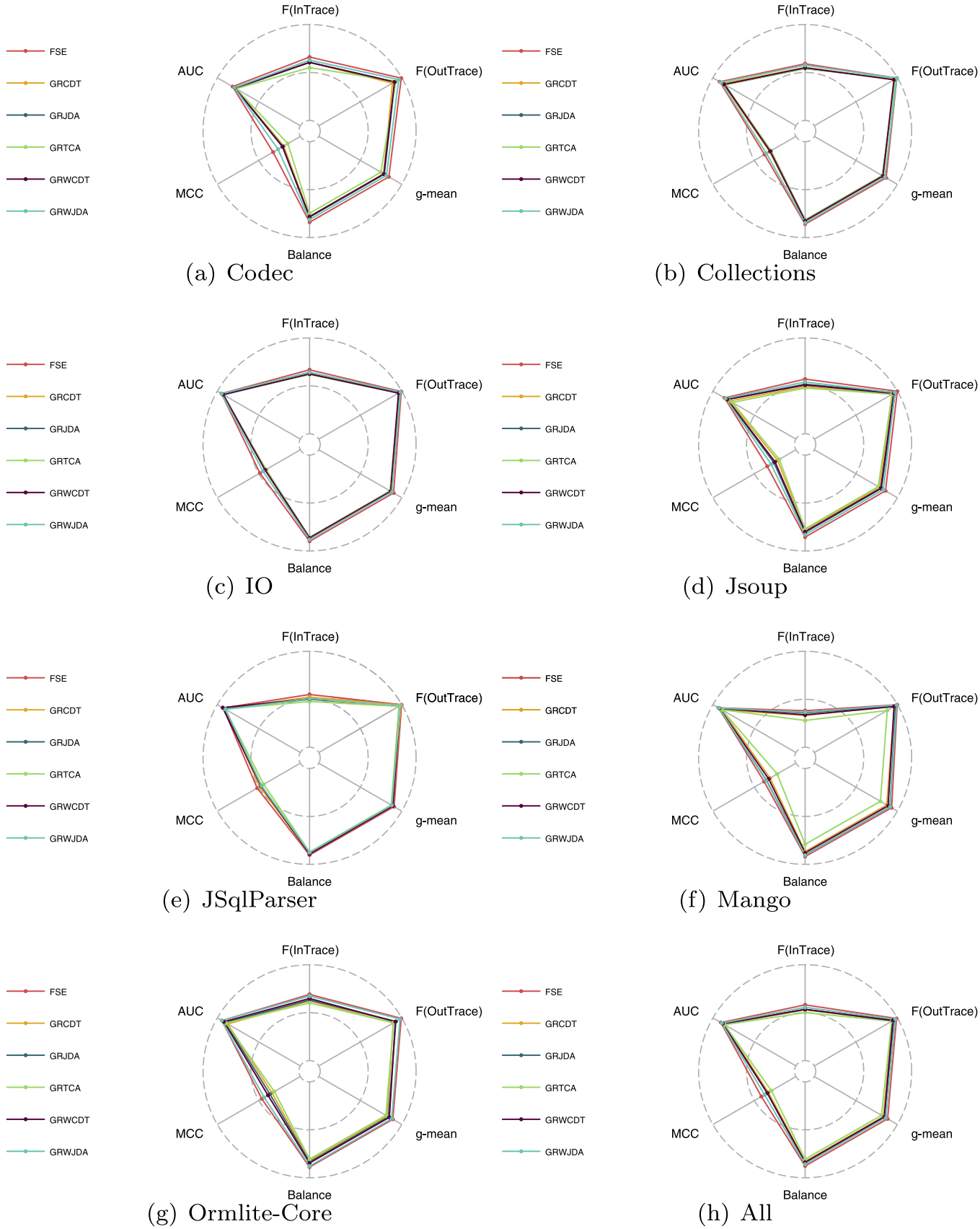

Fig. 7. The radar chart of average indicator values for FSE framework and 5 downgraded methods on each project and all projects.

feature embedding method **CDT** (**C**onditional **D**istribution based **T**ransfer learning) that only concerns about decreasing the conditional distribution discrepancy (i.e., $\mu = 1$ in Eq. (7)).

GRJDA method combines GR based feature selection with the feature embedding method **JDA** (**J**oint **D**istribution **A**nalysis) [55] that considers simultaneously the two distributions with the same weight (i.e., $\mu = 0.5$ in Eq. (7)).

GRWCDT method combines GR based feature selection with the **W**eighted **CDT** based feature embedding method that considers the class imbalanced issue when calculating the conditional distribution discrepancy for CDT.

GRWJDA method combines GR based feature selection with **W**eighted **JDA** based feature embedding method that considers the class imbalanced issue when calculating the conditional distribution discrepancy for JDA.

Note that, the weighted TCA method has the same results as TCA because the weighting scheme works only in the calculation process of the conditional distribution discrepancy while TCA method only involves in calculating marginal distribution discrepancy.

Results: Fig. 7 shows the radar chart of the 6 average indicator values for our proposed FSE framework and the 5 downgraded methods. From the figure, we have the following findings:

First, from Fig. 7(h), we find that our FSE framework achieves the best average values on all indicators across 42 cross project pairs compared with the 5 downgraded methods. More specifically, compared with the best average indicator values among the 5 downgraded methods, our FSE framework achieves the average improvements of 4.6%, 1.6%, 2.4%, 2.3%, 10.1%, and 1.2% in terms of F(InTrace), F(OutTrace), g-mean, Balance, MCC, and AUC, respectively.

Second, from Fig. 7(a) to (g), the results show that 5 average indicator values except in terms of AUC by our FSE framework are better than that by the 5 downgraded methods on all projects; from Fig. 7(e)–(g), the results show that average AUC value by our FSE framework is only a bit lower than that by one downgraded method on project JSqlParser, Mango, and Ormlite.

Third, Fig. 8 visualizes the corresponding statistic test results for the 6 methods in terms of the 6 indicators. The p-values show that there exist statistically significant differences among the 6 methods in terms of all indicators. The CD diagrams show that our FSE framework belongs to the top ranking group in terms of all indicators and achieves the best average ranking on all indicators. Our FSE framework has significant performance differences compared with all downgraded methods in terms of F(InTrace), g-mean, Balance, and MCC, whereas has no significant differences compared with 1 and 2 downgraded methods in terms of F(OutTrace) and AUC, respectively.

Analysis: Overall, the performance values of our FSE framework are better than that of GRWJDA, GRWCDT and GRTCA methods. This indicates that considering two distribution discrepancies and their different importance degrees on distinct cross project pairs is effective to achieve better ICFR performance. In other words, the similarity degrees indeed vary towards different cross project pairs and the two probability distributions of the cross project data should be treated differently. From the point of the average ranking in the CD diagrams, GRWJDA and GRWCDT methods perform better than GRJDA and GRCDT methods, respectively. This indicates that the weighting scheme is necessary to improve the cross project ICFR performance. In addition, GRWJDA and GRJDA method perform better than GRWCDT and GRCDT methods respectively, and GRWCDT and GRCDT methods perform better than GRTCA method. This indicates that the method considering two distribution discrepancies with equal weights is more effective than the one considering the conditional distribution discrepancy, and the method considering conditional distribution discrepancy is better than the one considering the marginal distribution discrepancy.

Answer: Considering both marginal and conditional distribution discrepancies with distinct weights is more effective to improve the cross project ICFR performance than the methods with equal weights or only considering one type of distribution discrepancy.

5.4. RQ4: Is our proposed FSE framework superior to the other cross project models?

Motivation: To the best of our knowledge, this is the first work to study the cross project ICFR task. Thus, we could not find the baseline methods tailored for ICFR to evaluate the effectiveness of our cross project ICFR framework. In this work, we select some cross project models for other learning tasks as our baseline methods.

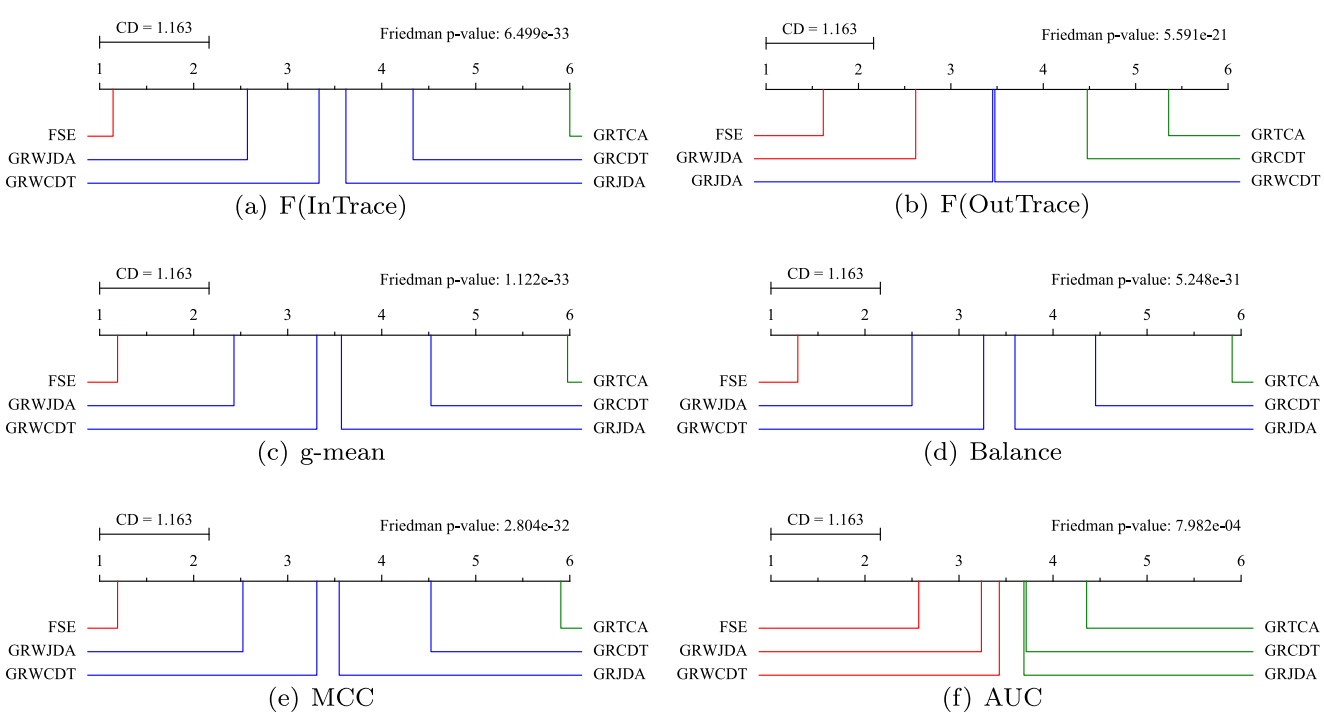

Fig. 8. Statistic test for FSE framework and 5 downgraded methods in terms of 6 indicators.

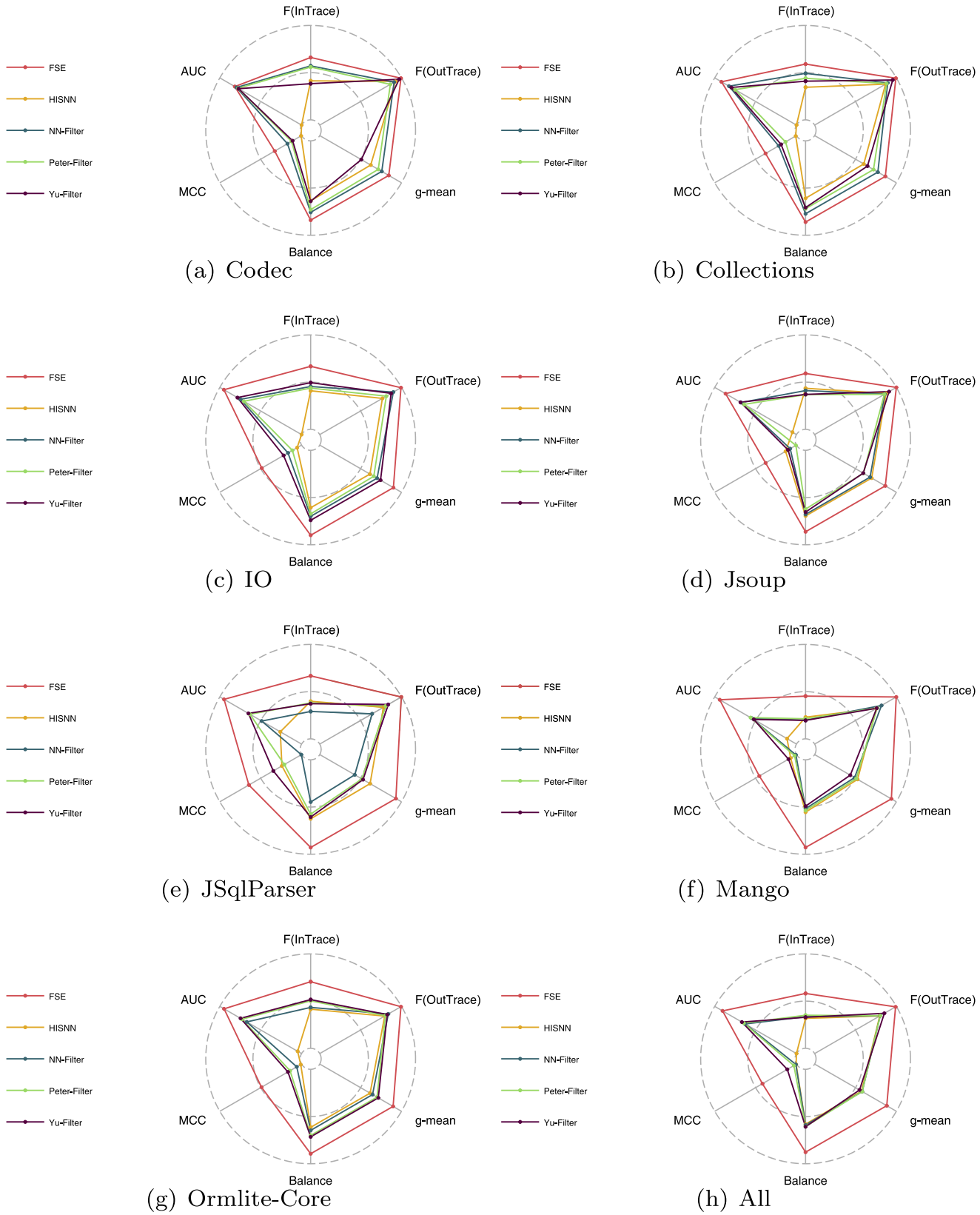

Fig. 9. The radar chart of average indicator values for FSE framework and 4 instance filtering based models on each project and all projects.

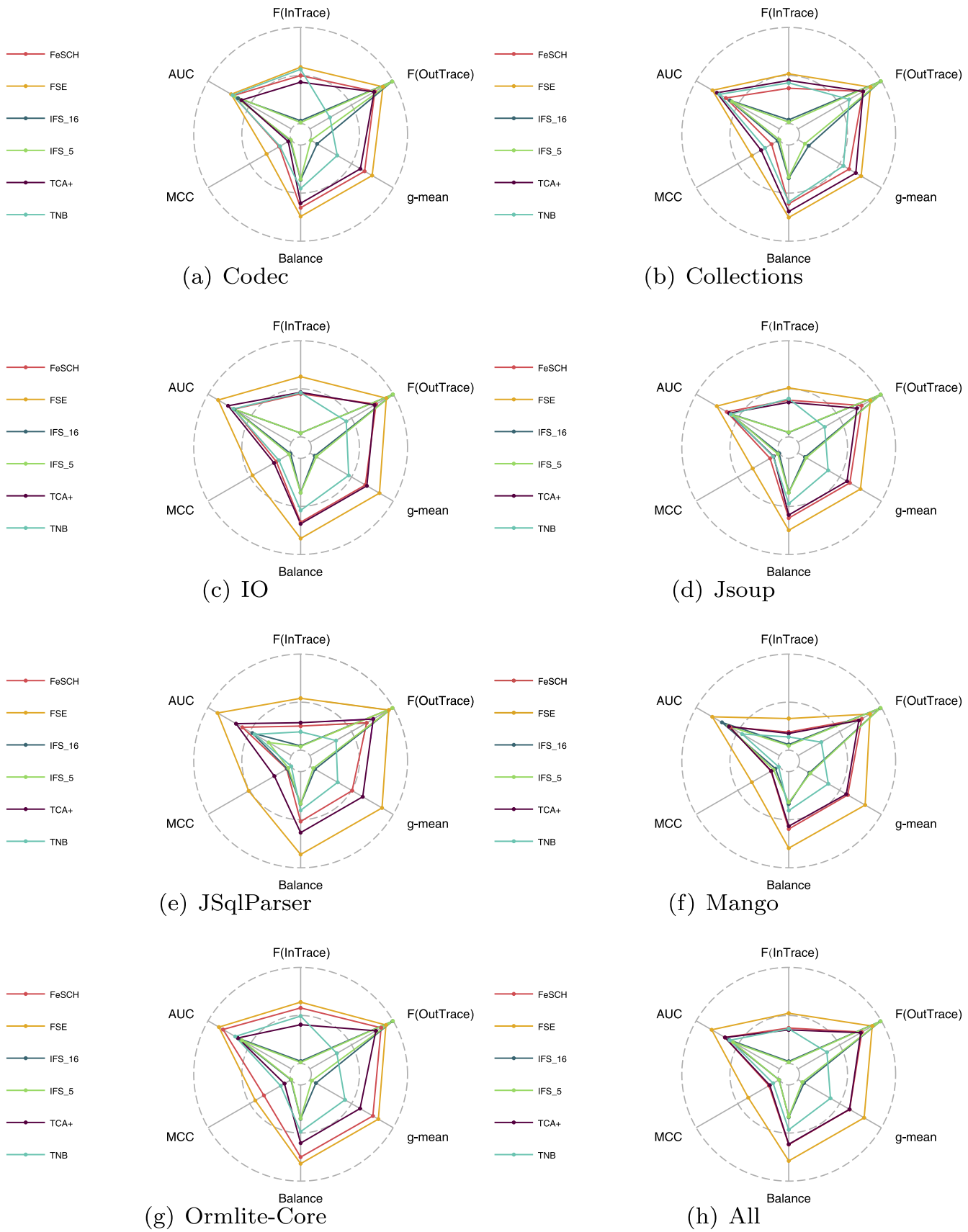

Fig. 10. The radar chart of average indicator values for FSE framework, 4 transfer learning based models, and one feature selection based model on each project and all projects.

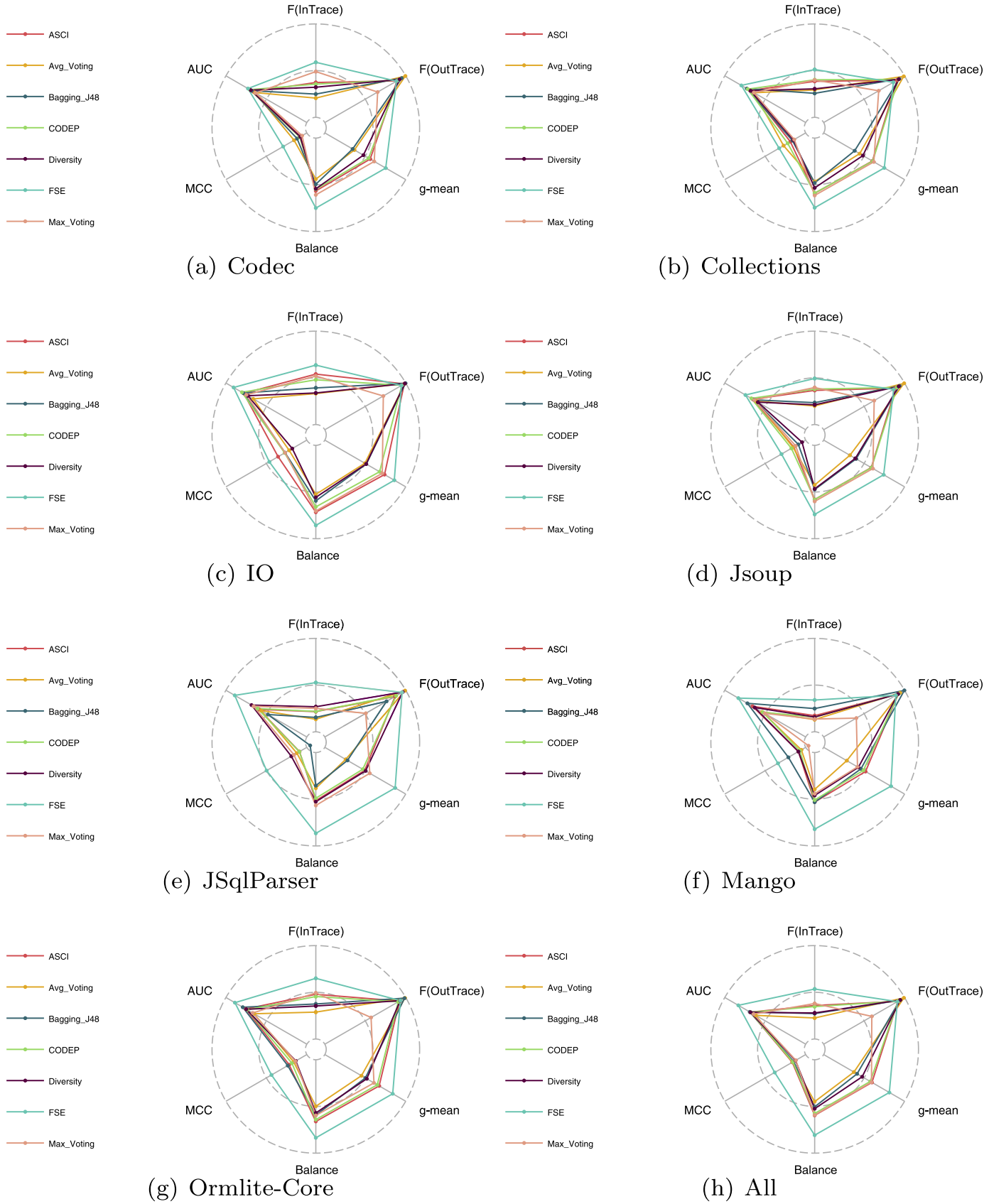

Fig. 11. The radar chart of average indicator values for FSE framework and 6 classifier combination based models on each project and all projects.

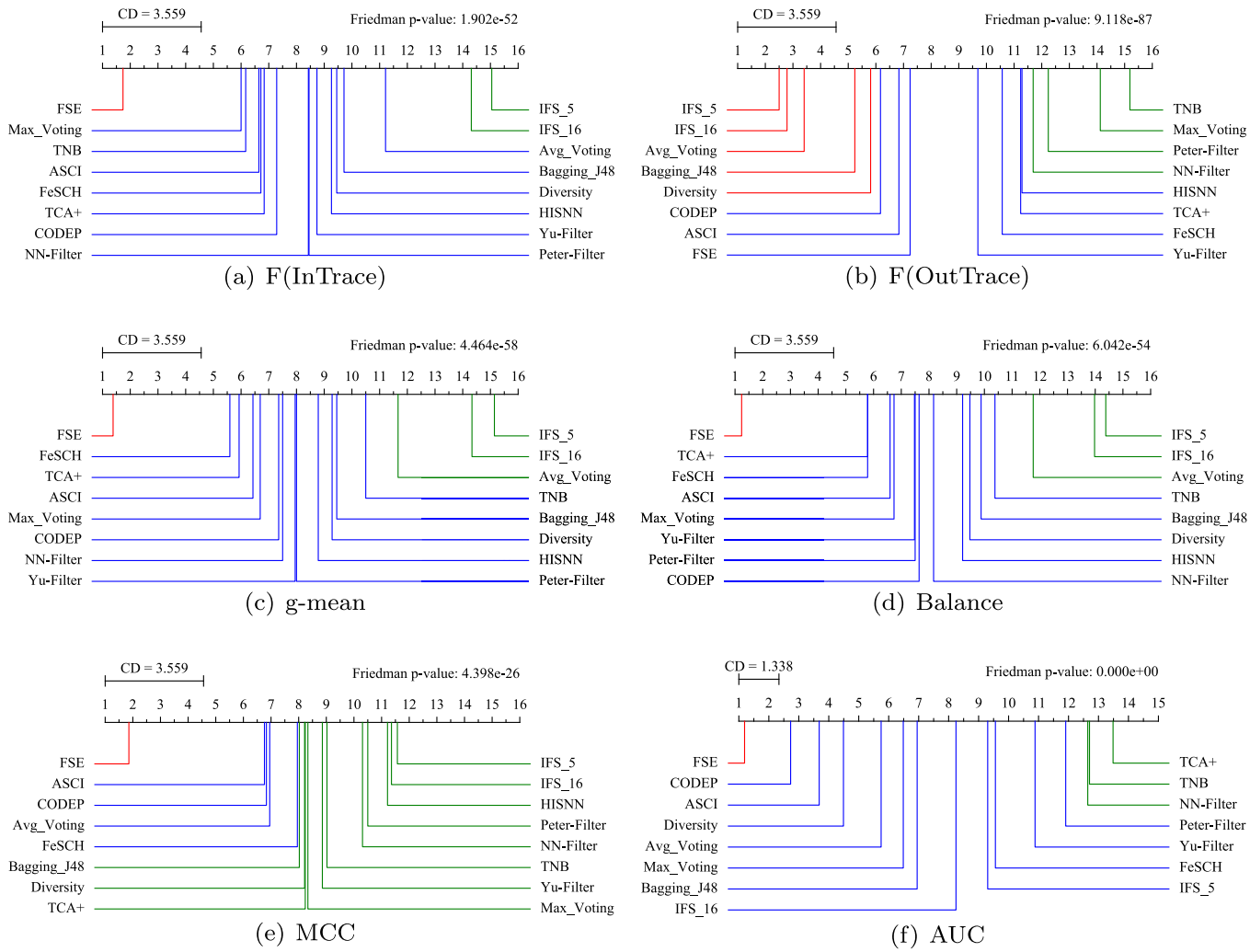

Fig. 12. Statistic test for FSE framework and 15 cross project models.

Method: We choose total 15 cross project models that are originally designed for software fault prediction task as our baseline methods, including 4 instance filtering based models (i.e., NN-Filter [28], Peter-Filter [29], Yu-Filter [30], and HISNN [65]), 4 transfer learning based models (i.e., TCA+ [15], TNB [32], IFS_5, and IFS_16 [66]), one feature selection based model (i.e., FeSCH[67]), and 6 classifier combination based models (i.e., Bagging_J48 [36], Max_Voting [36], Ave_-Voting [36], Diversity [37], CODEP [34], and ASCI [35]).

Results: Fig. 9 shows the radar chart of the 6 average indicator values for our FSE framework and 4 instance filtering based models. Fig. 10 shows the radar chart of the 6 average indicator values for our FSE framework, 4 transfer learning based models, and one feature selection based model. Fig. 11 shows the radar chart of the 6 average indicator values for our FSE framework and 6 classifier combination based models. From these figures, we have the following observations:

First, from Subfigure 9(h), we see that our FSE framework achieves the best average values on all indicators across 42 cross project pairs compared with 4 instance filtering based models. From Figs. 10(h) and 11(h), we see that our FSE framework achieves the best average values on 5 indicators except in terms of F(OutTrace). More specifically, compared with the best average indicator values among the 15 cross project models, our FSE framework achieves the average improvements of 41.4%, 28.1%, 27.4%, 119.4%, and 20.4% in terms of F(InTrace), g-mean, Balance, MCC, and AUC, respectively.

Second, from Fig. 9(a) to (g), the results show that all average indicator values by our FSE framework are better than the 4 instance filtering based models on all projects; from Figs. 10(a) to (g), and 11(a) to (g), the results show that 5 average indicator values except in terms of F(OutTrace) by our FSE framework are better than the 4 transfer learning based models, one feature selection based model, and 6 classifier combination based models on all projects.

Third, Fig. 12 visualizes the corresponding statistic test results for the 16 methods in terms of the 6 indicators. The p-values show that the 16 methods exist statistically significant performance differences in terms of all indicators. The CD diagrams show that our FSE framework belongs to the top ranking group in terms of 5 indicators except in terms of F (OutTrace) and achieves the best average ranking on the 5 indicators. In addition, our FSE framework has significant performance differences compared with all 15 cross project models in terms of 5 indicators except in terms of F(OutTrace), whereas 5 cross project models perform significantly better than our FSE framework in terms of F(OutTrace).

Analysis: The performance superior of our FSE framework is obvious compared with the 15 cross project models except in terms of F(OutTrace), which indicates that our proposed FSE framework is more suitable for cross project ICFR task. In addition, Yu-Filter model achieves the better overall performance among the 4 instance filtering based models; TCA+ model achieves the better overall performance among the 4 transfer learning based models; Max_Voting, ASCI, and CODEP models achieve the better overall performance among the 6 classifier combination based models; and feature selection FeSCH based model also achieves better performance. In addition, the average F(OutTrace) value by the FSE framework is lower than 2 transfer learning based models and

4 classifier combination based models. The reason is that our proposed FSE method uses a weighting strategy to deal with the class imbalance issue which will lead to that the results tend to be biased towards F (InTrace) and sacrifice some of the performance of F(OutTrace).

Answer: Our proposed FSE framework outperforms the comparative models that are designed for other cross project learning task, but there is still improvement room in terms of F(OutTrace) performance for our FSE framework since it is not always the best.

6. Threats to validity

6.1. External validity

We conduct experiments on a benchmark dataset that has been released recently. Since all the employed 7 projects are developed with Java language, future studies are necessary to investigate whether our results can be generalized to the projects developed with other languages. Although the used benchmark dataset is collected via imitating the crashes caused by the seeded faults using program mutation operations, previous studies have stated that the faults by mutation can be used as a replacement for the real-world faults as they share similar properties due to the fact that mutation satisfies the major principles of experimental design [68,69]. For the fault analysis task similar to our work, previous studies have shown that the detection of faults by mutation has a statistically significant correlation with the real-world fault [70], and the fault localization performance on faults by mutation has no statistical significance than that on real-world faults [71]. Thus, using the simulated data by mutation test to study the ICFR problem is an acceptable choice in the present case and the threat of using the simulative crashes is minor for the generalization ability of our results. In addition, in this work, we just use a simple feature selection method in our proposed framework as we just aim to investigate whether the feature selection process can promote the performance of cross project ICFR task. Other state-of-the-art feature selection techniques can also be considered in future studies.

6.2. Internal validity

The internal validity is threatened by the re-implementation of the baseline methods. Since the source code of most cross project models used in our experiment comparison is not provided by the authors, we carefully implement them following their details described in the corresponding studies to minimize the potential faults. We make our source code and benchmark dataset online, which allows further studies to replicate our experiments and confirm our results.

6.3. Construct validity

As single performance evaluation could potentially threaten the construct validity, in this work, we employ 6 indicators as performance measurements, which enables us to have a more comprehensive evaluation on the effectiveness of our method. In addition, we employ an improved statistic test to analyze the results, which makes our evaluation more convincing.

7. Conclusion

Due to the potential issues induced by the crash, the analysis and handling of the crash cannot be ignored. Designing method to automatically determine the localization of the crashing fault can assist developers for manual crash localization. The ICFR method has the potential to tell us specifically the localization where the crash is generated if the crash residence is predicted in the stack trace. For example, if the faulty code of a crash exactly matches the recorded information of one frame in the stack trace, the ICFR method will predict that the crashing fault is in the stack trace. In this case, the developers only need to inspect the limited code lines record in all frames for finding the faulty code. This will greatly reduce the search space for crashing faults. Thus, the ICFR method serves as a lightweight method to assist the fault localization task and help prioritize corresponding efforts. The typical usage scenario of our cross project ICFR model is to provide an early warning to developers about whether the crashing fault resides in the stack trace or not once the software crashes even no previous labeled crash data available. Our model provides this kind of early warning through the classification model built on the labeled data of other projects. More specifically, for an ongoing project, historical software development data may not be available to collect the residence information of the crashing fault as a training set for the ICFR task on forthcoming crashes. To resolve the dilemma of label shortage for ICFR task, we propose a novel two-stage cross project ICRF framework, called FSE, that consists of feature selection stage and feature embedding stage. In the first stage, our FSE framework uses a simple GR based feature ranking method to identify the relevant features and remove the irrelevant ones, then, it employs an advanced WBDA based feature embedding method to transform the feature space. WBDA has the advantage of reducing data distribution discrepancies of two projects and alleviating the class imbalance issue simultaneously. Using our FSE framework, the crashing fault of a new project that locates in the stack trace can be detected in an early stage by transferring the knowledge of the labeled crash data of other project. Then, only a few lines of code in the stack trace are needed for carefully inspection to find the root cause of the crash. This process can facilitate developers to fix the crash, accelerating the development process and saving debugging cost. The experiments on 7 open-source Java projects show that our proposed FSE framework achieves better prediction performance for the ICFR task than 25 baseline methods on 6 indicators overall.

In future, we plan to explore a feasible way to automatically specify the weights of two distribution differences for WBDA method and integrate new feature selection methods into our framework.

CRediT authorship contribution statement

Zhou Xu: Writing - review & editing, Methodology, Software, Data curation. **Tao Zhang:** Conceptualization, Visualization. **Jacky Keung:** Conceptualization, Writing - review & editing. **Meng Yan:** Supervision. **Xiapu Luo:** Supervision. **Xiaohong Zhang:** Project administration. **Ling Xu:** Formal analysis. **Yutian Tang:** Writing - review & editing.

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.

Acknowledgments

This work is supported by the National Key Research and Development Project (No. 2018YFB2101200), the National Natural Science Foundation of China (No. 62002034), China Postdoctoral Science Foundation (No. 2020M673137, No. 2017M621247), the Natural Science Foundation of Chongqing in China (No. cstc2020jcyj-bshX0114), the Science and Technology Development Fund of Macau (No. 0047/2020/A1), Faculty Research Grant Projects of MUST (No. FRG-20-008-FI), Hong Kong Research Grant Council Project (No. 152239/18E), the General Research Fund of the Research Grant Council of Hong Kong (No. 11208017), the Fundamental Research Funds for the Central Universities (No. 2020CDJQY-A021, No. 2019CDYGYB014).

References

[1] M. Hamill, K. Goseva-Popstojanova, Common trends in software fault and failure data, Trans. Softw. Eng. (TSE) 35 (4) (2009) 484–496.

[2] D. Cotroneo, L. De Simone, P. Liguori, R. Natella, N. Bidokhti, How bad can a bug get? An empirical analysis of software failures in the OpenStack cloud computing platform. Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE), ACM, 2019, pp. 200–211.

[3] W.E. Wong, R. Gao, Y. Li, R. Abreu, F. Wotawa, A survey on software fault localization, Trans. Softw. Eng. (TSE) 42 (8) (2016) 707–740.

[4] T. Zhang, J. Chen, X. Luo, T. Li, Bug reports for desktop software and mobile apps in GitHub: what's the difference? IEEE Softw. 36 (1) (2017) 63–71.

[5] Y. Gu, J. Xuan, H. Zhang, L. Zhang, Q. Fan, X. Xie, T. Qian, Does the fault reside in a stack trace? Assisting crash localization by predicting crashing fault residence, J. Syst. Softw. (JSS) 148 (2019) 88–104.

[6] L. Song, L.L. Minku, X. Yao, A novel automated approach for software effort estimation based on data augmentation. Proceedings of the 26th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE), 2018, pp. 468–479.

[7] J. Wang, Y. Chen, S. Hao, W. Feng, Z. Shen, Balanced distribution adaptation for transfer learning. Proceedings of the 17th International Conference on Data Mining (ICDM), 2017, pp. 1129–1134.

[8] M. Bilgic, Combining active learning and dynamic dimensionality reduction. Proceedings of the 2012 SIAM International Conference on Data Mining, SIAM, 2012, pp. 696–707.

[9] Z. Xu, J. Xuan, J. Liu, X. Cui, MICHAC: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER) vol. 1, IEEE, 2016, pp. 370–381.

[10] S. Shivaji, E.J. Whitehead Jr, R. Akella, S. Kim, Reducing features to improve bug prediction. 2009 IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2009, pp. 600–604.

[11] S. Shivaji, E.J. Whitehead, R. Akella, S. Kim, Reducing features to improve code change-based bug prediction, IEEE Trans. Softw. Eng. (TSE) 39 (4) (2012) 552–569.

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

[13] F. Zhang, I. Keivanloo, Y. Zou, Data transformation in cross-project defect prediction, Empir. Softw. Eng. (EMSE) 22 (6) (2017) 3186–3218.

[14] C. Liu, D. Yang, X. Xia, M. Yan, X. Zhang, A two-phase transfer learning model for cross-project defect prediction, Inf. Softw. Technol. (IST) (2018) 125–136.

[15] J. Nam, S.J. Pan, S. Kim, Transfer defect learning. Proceedings of the 35th International Conference on Software Engineering (ICSE), 2013, pp. 382–391.

[16] Z. Xu, T. Zhang, Y. Zhang, Y. Tang, J. Liu, X. Luo, J. Keung, X. Cui, Identifying crashing fault residence based on cross project model. Proceedings of the 30th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2019.

[17] J. Xuan, X. Xie, M. Monperrus, Crash reproduction via test case mutation: let existing test cases help. Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (FSE), 2015, pp. 910–913.

[18] Y. Dang, R. Wu, H. Zhang, D. Zhang, P. Nobel, ReBucket: a method for clustering duplicate crash reports based on call stack similarity. Proceedings of the 34th International Conference on Software Engineering (ICSE), 2012, pp. 1084–1093.

[19] T. Dhaliwal, F. Khomh, Y. Zou, Classifying field crash reports for fixing bugs: a case study of Mozilla Firefox. Proceedings of the 27th International Conference on Software Maintenance (ICSM), 2011, pp. 333–342.

[20] N. Chen, S. Kim, STAR: Stack trace based automatic crash reproduction via symbolic execution, Trans. Softw. Eng. (TSE) 41 (2) (2015) 198–220.

[21] M. Nayrolles, A. Hamou-Lhadj, S. Tahar, A. Larsson, A bug reproduction approach based on directed model checking and crash traces, J. Softw. 29 (3) (2017) e1789.

[22] M. Nayrolles, A. Hamou-Lhadj, S. Tahar, A. Larsson, JCHARMING: A bug reproduction approach using crash traces and directed model checking. Proceedings of the 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER), 2015, pp. 101–110.

[23] M. Soltani, A. Panichella, A. Van Deursen, A guided genetic algorithm for automated crash reproduction. Proceedings of the 39th International Conference on Software Engineering, IEEE Press, 2017, pp. 209–220.

[24] R. Wu, H. Zhang, S.-C. Cheung, S. Kim, CrashLocator: locating crashing faults based on crash stacks. Proceedings of the 23rd International Symposium on Software Testing and Analysis (ISSTA), 2014, pp. 204–214.

[25] S. Wang, F. Khomh, Y. Zou, Improving bug localization using correlations in crash reports. Proceedings of the 10th Working Conference on Mining Software Repositories (MSR), 2013, pp. 247–256.

[26] L. Moreno, J.J. Treadway, A. Marcus, W. Shen, On the use of stack traces to improve text retrieval-based bug localization. Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME), 2014, pp. 151–160.

[27] C.-P. Wong, Y. Xiong, H. Zhang, D. Hao, L. Zhang, H. Mei, Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. Proceedings of the 30th International Conference on Software Maintenance and Evolution (ICSME), 2014, pp. 181–190.

[28] B. Turhan, T. Menzies, A.B. Bener, J. Di Stefano, On the relative value of cross-company and within-company data for defect prediction, Empir. Softw. Eng. (EMSE) 14 (5) (2009) 540–578.

[29] F. Peters, T. Menzies, A. Marcus, Better cross company defect prediction. Proceedings of the 10th Working Conference on Mining Software Repositories (MSR), 2013, pp. 409–418.

[30] X. Yu, J. Zhang, P. Zhou, J. Liu, A data filtering method based on agglomerative clustering. Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2017, pp. 392–397.

[31] K. Kawata, S. Amasaki, T. Yokogawa, Improving relevancy filter methods for cross-project defect prediction. Proceedings of the 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence, 2015, pp. 2–7.

[32] Y. Ma, G. Luo, X. Zeng, A. Chen, Transfer learning for cross-company software defect prediction, Inf. Softw. Technol. (IST) 54 (3) (2012) 248–256.

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

[34] A. Panichella, R. Oliveto, A. De Lucia, Cross-project defect prediction models: L'union fait la force. Proceedings of the 21st Software Evolution Week Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014, pp. 164–173.

[35] D. Di Nucci, F. Palomba, A. De Lucia, Evaluating the adaptive selection of classifiers for cross-project bug prediction. Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, IEEE, 2018, pp. 48–54.

[36] Y. Zhang, D. Lo, X. Xia, J. Sun, An empirical study of classifier combination for cross-project defect prediction. Proceedings of the 39th Annual Computer Software and Applications Conference (COMPSAC) vol. 2, 2015, pp. 264–269.

[37] J. Petrić, D. Bowes, T. Hall, B. Christianson, N. Baddoo, Building an ensemble for software defect prediction based on diversity selection. Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2016, pp. 1–10.

[38] S. Herbold, A. Trautsch, J. Grabowski, A comparative study to benchmark cross-project defect prediction approaches, IEEE Trans. Softw. Eng. (TSE) 44 (9) (2017) 811–833.

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

[40] E. Kocaguneli, G. Gay, T. Menzies, Y. Yang, J.W. Keung, When to use data from other projects for effort estimation. Proceedings of the 25th International Conference on Automated Software Engineering (ASE), 2010, pp. 321–324.

[41] L.L. Minku, X. Yao, How to make best use of cross-company data in software effort estimation?. Proceedings of the 36th International Conference on Software Engineering (ICSE), 2014, pp. 446–456.

[42] Y. Ge, M. Chen, C. Liu, F. Chen, S. Huang, H. Wang, Deep metric learning for software change-proneness prediction. Proceedings of International Conference on Intelligent Science and Big Data Engineering, 2018, pp. 287–300.

[43] A. Bansal, S. Jajoria, Cross-project change prediction using meta-heuristic techniques, Int. J. Appl. Metaheuristic Comput. 10 (1) (2019) 43–61.

[44] L. Chao, Y. Dan, X. Xin, Y. Meng, X. Zhang, Cross-project change-proneness prediction. Proceedings of the 42th Annual Computer Software and Applications Conference (COMPSAC) vol. 1, 2018, pp. 64–73.

[45] S. Lal, N. Sardana, A. Sureka, Three-level learning for improving cross-project logging prediction for if-blocks, J. King Saud Univ.-Comput.Inf. Sci. (2017) 1–16.

[46] S. Lal, N. Sardana, A. Sureka, ECLogger: cross-project catch-block logging prediction using ensemble of classifiers, e-Inf. Softw. Eng. J. 11 (1) (2017).

[47] A. Takanen, J.D. Demott, C. Miller, A. Kettunen, Fuzzing for Software Security Testing and Quality Assurance, Artech House, 2018.

[48] M. Zhao, P. Liu, Empirical analysis and modeling of black-box mutational fuzzing. Proceedings of the 8th International Symposium on Engineering Secure Software and Systems, 2016, pp. 173–189.

[49] H. Shahriar, S. North, E. Mawangi, Testing of memory leak in android applications. Proceedings of the 15th International IEEE Symposium on High-Assurance Systems Engineering (HASE), 2014, pp. 176–183.

[50] Z. Michal, American fuzzy lop, 2016, http://lcamtuf.coredump.cx/afl.

[51] M. Pudas, Improving crash uniqueness detection in fuzzy testing: case jyvsectec (2017).

[52] H. Chen, Y. Xue, Y. Li, B. Chen, X. Xie, X. Wu, Y. Liu, Hawkeye: towards a desired directed grey-box fuzzer. Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS), 2018, pp. 2095–2108.

[53] J. Liang, M. Wang, Y. Chen, Y. Jiang, R. Zhang, Fuzz testing in practice: obstacles and solutions. Proceedings of the IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2018, pp. 562–566.

[54] B. Zhang, C. Feng, A. Herrera, V. Chipounov, G. Candea, C. Tang, Discover deeper bugs with dynamic symbolic execution and coverage-based fuzz testing, IET Softw. 12 (6) (2018) 507–519.

[55] M. Long, J. Wang, G. Ding, J. Sun, P.S. Yu, Transfer feature learning with joint distribution adaptation. Proceedings of the 14th International Conference on Computer Vision (ICCV), 2013, pp. 2200–2207.

[56] S.J. Pan, I.W. Tsang, J.T. Kwok, Q. Yang, Domain adaptation via transfer component analysis, Trans. Neural Netw. (TNN) 22 (2) (2011) 199–210.

[57] Z. Lin, M. Chen, Y. Ma, The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices, arXiv:1009.5055 (2010).

[58] Z. Xu, J. Liu, X. Luo, Z. Yang, Y. Zhang, P. Yuan, Y. Tang, T. Zhang, Software defect prediction based on kernel PCA and weighted extreme learning machine, Inf. Softw. Technol. (IST) 106 (2019) 182–200.

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

[60] Y. Yang, M. Harman, J. Krinke, S. Islam, D. Binkley, Y. Zhou, B. Xu, An empirical study on dependence clusters for effort-aware fault-proneness prediction. 2016

31st IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2016, pp. 296–307.
[61] J. Nam, W. Fu, S. Kim, T. Menzies, L. Tan, Heterogeneous defect prediction, IEEE Trans. Softw. Eng. (TSE) 44 (9) (2017) 874–896.
[62] X. Xia, D. Lo, S.J. Pan, N. Nagappan, X. Wang, HYDRA: Massively compositional model for cross-project defect prediction, IEEE Trans. Softw. Eng. (TSE) 42 (10) (2016) 977–998.
[63] X. Jing, F. Wu, X. Dong, F. Qi, B. Xu, Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 496–507.
[64] Z. Xu, P. Yuan, T. Zhang, Y. Tang, S. Li, Z. Xia, HDA: Cross-project defect prediction via heterogeneous domain adaptation with dictionary learning, IEEE Access 6 (2018) 57597–57613.
[65] D. Ryu, J.-I. Jang, J. Baik, A hybrid instance selection using nearest-neighbor for cross-project defect prediction, J. Comput. Sci. Technol. (JCST) 30 (5) (2015) 969–980.
[66] P. He, B. Li, Y. Ma, Towards cross-project defect prediction with imbalanced feature sets, arXiv:1411.4228 (2014).
[67] C. Ni, W.-S. Liu, X. Chen, Q. Gu, D.-X. Chen, Q.-G. Huang, A cluster based feature selection method for cross-project software defect prediction, J. Comput. Sci. Technol. (JCST) 32 (6) (2017) 1090–1107.
[68] J.H. Andrews, L.C. Briand, Y. Labiche, Is mutation an appropriate tool for testing experiments. Proceedings of the 27th International Conference on Software Engineering (ICSE), 2005, pp. 402–411.
[69] A.S. Namin, S. Kakarla, The use of mutation in testing experiments and its sensitivity to external threats. Proceedings of the 20th International Symposium on Software Testing and Analysis (ISSTA), 2011, pp. 342–352.
[70] R. Just, D. Jalali, L. Inozemtseva, M.D. Ernst, R. Holmes, G. Fraser, Are mutants a valid substitute for real faults in software testing. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), 2014, pp. 654–665.
[71] S. Ali, J.H. Andrews, T. Dhandapani, W. Wang, Evaluating the accuracy of fault localization techniques. Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2009, pp. 76–87.