Identifying Crashing Fault Residence Based on Cross Project Model

Zhou Xu[a,b], Tao Zhang[c], Yifeng Zhang[a], Yutian Tang[b], Jin Liu[a,d,*], Xiapu Luo[b,*], Jacky Keung[e], and Xiaohui Cui[f]

[a]School of Computer Science, Wuhan University, China
[b]Department of Computing, The Hong Kong Polytechnic University, China
[c]College of Computer Science and Technology, Harbin Engineering University, China
[d]Key Laboratory of Network Assessment Technology, Institute of Information Engineering, Chinese Academy of Sciences, China
[e]Department of Computer Science, City University of Hong Kong, China
[f]School of Cyber Science and Engineering, Wuhan University, China

***Abstract*—Analyzing the crash reports recorded upon software crashes is a critical activity for software quality assurance. Predicting whether or not the fault causing the crash (crashing fault for short) resides in the stack traces of crash reports can speed-up the program debugging process and determine the priority of the debugging efforts. Previous work mostly collected label information from bug-fixing logs, and extracted crash features from stack traces and source code to train classification models for the Identification of Crashing Fault Residence (ICFR) of newly-submitted crashes. However, labeled data are not always fully available in real applications. Hence the classifier training is not always feasible. In this work, we make the first attempt to develop a cross project ICFR model to address the data scarcity problem. This is achieved by transferring the knowledge from external projects to the current project via utilizing a state-of-the-art Balanced Distribution Adaptation (BDA) based transfer learning method. BDA not only combines both marginal distribution and conditional distribution across projects but also assigns adaptive weights to the two distributions for better adjusting specific cross project pair. The experiments on 7 software projects show that BDA is superior to 9 baseline methods in terms of 6 indicators overall.**

***Index Terms*—crashing fault, stack trace, transfer learning, cross project model**

I. Introduction

Due to the complex program structure, software products may well contain faults (or bugs) upon release. The faults have high risks to activate the 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) [1].

Generally, crash localization utilizes the information of the stack trace and the source code to find the root cause of the crash for debugging. 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 inspect extensive source code [2]. This will seriously hinder the efficiency demand of the crash localization.

To facilitate the crash localization, Gu et al. [2] 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 in the within-project scenario, in which 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 [3]–[5], 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 the insufficiently labeled or unlabeled data of the project at hand (*a.k.a.* target project). As far as we know, the study of ICFR in cross project setting is uninvestigated yet. In this paper, we make the first attempt on this problem by proposing a cross project ICFR model.

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). In this work, we apply a novel transfer learning method, called **B**alanced **D**istribution **A**daptation (**BDA**) [6], as our cross project ICFR model. The advantage of BDA is that it not only considers discrepancies of both marginal and conditional probability distributions, but also allocates different weights

*Corresponding authors: jinliu@whu.edu.cn, csxluo@comp.polyu.edu.hk

978-1-7281-4982-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ISSRE.2019.00027

IEEE computer society

Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.

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, while if the data distribution of two projects is dissimilar, the marginal probability distribution has higher importance [6].

In addition, since 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 [7]. 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 [8]. To alleviate this issue, Nam et al. [9] 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 ICFR data, and investigate whether or not the normalization adaption selection in [9] also works well on our cross project ICFR model.

We conduct experiments on a public dataset collecting from 7 open-source projects, and employ 6 indicators to evaluate the ICFR performance of our BDA based cross project model. The detailed results show that BDA achieves 0.104∼0.192 improvements in terms of F-measure for crash instances inside the stack trace, 0.132∼0.394 improvements in terms of G-measure, 0.127∼0.334 improvements in terms of g-mean, 0.124∼0.263 improvements in terms of Balance, 0.137∼0.296 improvements in terms of MCC, all over 9 baseline methods. In addition, ICFR outperforms 7 out of 9 baseline methods in terms of F-measure for crash instances outside the stack trace.

In summary, we have the following contributions:

(1) In this work, we are among the first to study the ICFR problem using the cross project model to address the data scarcity issue in which the labeled data are not always available for the project at hand.
(2) By leveraging a state-of-the-art BDA method, we alleviate the threat of distribution inconsistency between cross project data to the performance of the cross project model. BDA is the first transfer learning technique that incorporates both marginal and conditional distributions together with adaptive weights.
(3) We comprehensively evaluate the proposed cross project model on 7 open-source projects with 6 indicators. The experimental results show the superiority of our devised cross project ICFR model over 9 baseline methods.

II. Related Work

A. 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 assists in reproducing the crash scenario, which is helpful in finding the root cause of the crash [10]. 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 least recent one is usually called bottom frame [2]. 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 [11], [12], crash reproduction [10], [13]–[15], crash localization [16]–[18], etc. Among these studies, crash localization is the most similar to our work.

Crash localization (or fault localization) recommends the developers a set of candidate functions based on their suspicious scores being faulty by analyzing the stack traces and source code. Schröter et al. [19] showed the strong evidence that stack traces are helpful for bug fixing as their initial results revealed that the faulty function is typically located in the top 10 stack frames. Wu et al. [16] 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. [20] 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. [17] proposed the Lobster method based on the structural and textual similarities. Wong et al. [18] developed a tool, called BRTracer, by analyzing the segmentation of the source code and stack traces for bug reports based fault localization. Jiang et al. [21] proposed a null pointer exception based fault localization method by analyzing the stack-trace-driven program slicing.

Different from the above fault localization studies which return the potential faulty functions to the developers, Gu et al. [2] proposed the CraTer method to determine whether the location of the faulty code is consistent with the record (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. [2] only considered ICFR in the same project setting, in this work, we extend their work to cross project setting since the labeled data are usually scarce in single project case.

B. Cross Project Learning Task

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, logging prediction, etc.

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

Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.

the target project are fault-prone. Prior studies have proposed different cross project models for this task, such as instance filtering based methods [22]–[25], transfer learning based methods [8], [9], [26], [27], and classifier combination based methods [28]–[31]. The studies of cross project fault prediction compose the most active research branch in software engineering field [32], [33].

Cross project effort estimation [34]–[39] estimates the effort required to develop a target project with the aid of the labeled data from other projects. Cross project change prediction [40]–[43] 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 [44]–[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 ICFR task.

III. Our Method

A. Notations

For 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. 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 i-th crash instance in X_S, y_s^i represents the corresponding label, d_s and n_s represent the number of features and crash instances, individually. 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 $X_T = x_t^i|_{i=1}^{n_t} \in \mathbb{R}^{n_t \times d_t}$, where x_t^i represents the i-th crash instance in X_T, d_t and n_t represent the number of features and crash instances, respectively. The corresponding label vector $Y_T = y_t^i|_{i=1}^{n_t}$ is what we are seeking for. Assume $\mathcal{X}_s$ ($\mathcal{X}_t$) 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 is to learn a common feature space in which the two distribution differences, 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.

B. Balanced Distribution Adaptation (BDA)

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

$$d(\mathcal{D}_S, \mathcal{D}_T) = d(\mathcal{P}(x_s), \mathcal{P}(x_t)) + d(\mathcal{P}(y_s|x_s), \mathcal{P}(y_t|x_t)). \quad (1)$$

For two data sets, the margin distribution is more important when they are dissimilar, otherwise the conditional distribution should be emphasized [6]. Thus just combining the two terms with the same weight in Eq.1 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:

$$d(\mathcal{D}_S, \mathcal{D}_T) = (1-\mu)d(\mathcal{P}(x_s), \mathcal{P}(x_t)) + \mu d(\mathcal{P}(y_s|x_s), \mathcal{P}(y_t|x_t)), \quad (2)$$

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 be calculated. Long et al. [47] 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 [48] 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. 2 is rewritten as

$$\begin{aligned} d(\mathcal{D}_S, \mathcal{D}_T) = (1-\mu) & \left\| \frac{1}{n_s} \sum_{i=1}^{n_s} x_s^i - \frac{1}{n_t} \sum_{j=1}^{n_t} x_t^j \right\|_{\mathcal{H}}^2 \\ & + \mu \sum_{c=1}^{C} \left\| \frac{1}{n_s^c} \sum_{x_s^i \in \mathcal{D}_S^{(c)}} x_s^i - \frac{1}{n_t^c} \sum_{x_t^j \in \mathcal{D}_T^{(c)}} x_t^j \right\|_{\mathcal{H}}^2, \end{aligned} \quad (3)$$

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. 3 is converted to

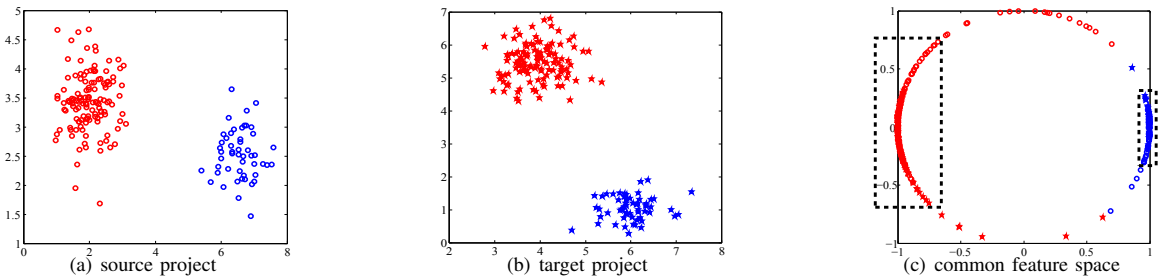
$$\begin{aligned} & \min_A \operatorname{tr}\Big(A^\top X \Big((1-\mu)M_0 + \mu \sum_{c=1}^{C} M_c\Big) X^\top A\Big) + \lambda \|A\|_F^2 \\ & \text{s.t. } A^\top X H X^\top A = I, 0 \leq \mu \leq 1, \end{aligned} \quad (4)$$

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 controls 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 $(n_s+n_t) \times (n_s+n_t)$, $H = I - (1/n)\mathbf{1}$ represents a centering matrix, and $\|A\|_F^2$ represents the Frobenius norm of A. M_0 and M_c represent MMD matrices as follows:

$$(M_0)_{ij} = \begin{cases} \frac{1}{n_s^2}, & x_i, x_j \in \mathcal{D}_S \\ \frac{1}{n_t^2}, & x_i, x_j \in \mathcal{D}_T \\ -\frac{1}{n_s n_t}, & \text{otherwise,} \end{cases} \quad (5)$$

$$(M_c)_{ij} = \begin{cases} \frac{1}{n_s^{c2}}, & x_i, x_j \in \mathcal{D}_S^{(c)} \\ \frac{1}{n_t^{c2}}, & x_i, x_j \in \mathcal{D}_T^{(c)} \\ -\frac{1}{n_s^c n_t^c}, & \begin{cases} x_i \in \mathcal{D}_S^{(c)}, x_j \in \mathcal{D}_T^{(c)} \\ x_i \in \mathcal{D}_T^{(c)}, x_j \in \mathcal{D}_S^{(c)} \end{cases} \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.



(a) source project (b) target project (c) common feature space

Fig. 1: An example of the feature transformation effect by BDA.

By using the Lagrange multiplier method [49], the Lagrange function of Eq. 4 is

$$L = \mathrm{tr}\Big(A^\top X\Big((1-\mu)M_0 + \mu\sum_{c=1}^{C} M_c\Big)X^\top A\Big) + \lambda\|A\|_F^2 + \mathrm{tr}((I - A^\top XHX^\top A)\Phi), \tag{7}$$

where $\Phi = (\phi_1, ..., \phi_d)$ represents the Lagrange multiplier. By setting the first-order derivative of L towards A to 0, Eq. 7 is converted to a generalized eigen-decomposition problem as

$$\Big(X\Big((1-\mu)M_0 + \mu\sum_{c=1}^{C} M_c\Big)X^\top + \lambda I\Big)A = XHX^\top A\Phi. \tag{8}$$

The result of Eq. 8 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 (green circle) from a mixture of Gaussian with means (6.5, 2.5), as showed in Figure 1(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 Figure 1(b). Figures 1(c) depicts the mapped data of two projects by BDA method with the equal weight in the common feature space. From Figure 1, 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 differences between the two projects.

After obtaining the mapped data of the two projects, we use the same logistic regression model in [9] as our basic classifier to conduct the ICFR task.

IV. Experimental Setup

Figure 2 provides an overview of the framework of this study. Below, we discuss the design components including the benchmark dataset and performance indicators.

A. Data Collection

In this work, we employ a publicly available benchmark dataset denoted by Gu et al. [2] to evaluate our proposed

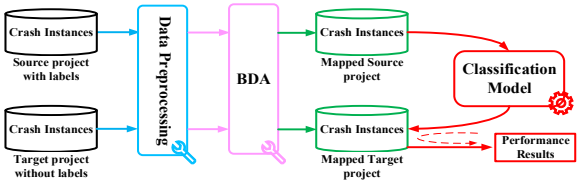


Fig. 2: An overview diagram of the experiment framework.

cross project ICFR model. 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 I 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 $\frac{\#\mathrm{OutTrace}}{\#\mathrm{InTrace}}$. The data collection consists of 3 main steps: crash generation, crashing fault residence labeling and feature extraction. We briefly describe each step as follows:

TABLE I: 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

1) Crash Generation:

(1) Fault Generation via Program Mutation

Since it is a non-trivial activity to reproduce the real-world crashes, Gu et al. [2] simulated the crashes by seeding faults into the real-world projects. More specifically, they applied a

Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.

state-of-the-art mutation testing tool, the PIT system[1], 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[2] 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.

2) Crashing Fault Residence 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 within the stack trace and labeled as 'InTrace', otherwise labeled as 'OutTrace'. The label information is collected by checking the bug-fixing logs from mutation testing.

3) Feature Extraction: In order to character each crash instance (i.e., crashing fault), Gu et al. [2] extracted 89 features in total from the stack trace and the relevant source code. These features come from 5 families as follows:

- 11 features based on the stack trace (marking from ST01 to ST11 for simplicity). These features represent the difficulty of handling the corresponding crash.
- 23 features based on the function and class in the top frame (marking from CT01 to CT23). As the position in which the exception is thrown is located in the top frame, the features of the function and class in the top frame can characterize the program state when it encounters a crash.
- 23 features based on the function and class in the bottom frame (marking from CB01 to CB23). As the information of initial function call is recorded in the bottom frame, the feature of the function and class in the bottom frame can also characterize the crashing fault.
- 16 features by normalizing CT08∼CT23 with LOC (marking from AT01 to AT16).
- 16 features by normalizing CB08∼CB23 with LOC (marking from AB01 to AB16).

Their brief descriptions are summarized in Table II.

B. 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. There are 4 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 $\hat{t}$ ($\hat{t}$ is the opposite of t),
- a crash with labeled $\hat{t}$ is predicted as $\hat{t}$,
- a crash with labeled $\hat{t}$ 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**).

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., $\mathrm{Precision(t)} = \frac{\mathrm{TP(t)}}{\mathrm{TP(t)+FP(t)}}$. 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., $\mathrm{PD(t)} = \mathrm{Recall(t)} = \frac{\mathrm{TP(t)}}{\mathrm{TP(t)+FN(t)}}$. 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., $\mathrm{PF(t)} = \frac{\mathrm{FP(t)}}{\mathrm{FP(t)+TN(t)}}$.

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

$$\text{F-measure(t)} = \frac{2 \times \mathrm{Precision(t)} \times \mathrm{Recall(t)}}{\mathrm{Precision(t)} + \mathrm{Recall(t)}} \tag{9}$$

G-measure is the harmonic mean of PD and (1-PF) as

$$\text{G-measure(t)} = \frac{2 \times \mathrm{PD(t)} \times (1 - \mathrm{PF(t)})}{\mathrm{PD(t)} + (1 - \mathrm{PF(t)})} \tag{10}$$

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

$$\text{g-mean(t)} = \sqrt{\mathrm{PD(t)} \times (1 - \mathrm{PF(t)})} \tag{11}$$

Balance is a trade-off between Recall and PF as

$$1 - \sqrt{\frac{(0 - \mathrm{PF(t)})^2 + (1 - \mathrm{Recall(t)})^2}{2}} \tag{12}$$

MCC is a correlation coefficient considering TP, TN, FP, and FN, which is defined as (omitting (t) for the 4 terms)

$$\mathrm{MCC(t)} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}} \tag{13}$$

In this study, we employ F-measure, G-measure, g-mean, Balance, and MCC as performance indicators. F-measure, G-measure, g-mean, and Balance range from 0 to 1, and MCC ranges from -1 to 1. Larger indicator value means better ICFR performance. In ICFR scenario, both the crashes with label 'InTrace' and 'OutTrace' can be treated as the positive instances. Thus for each indicator, we need to calculate two performance values which correspond to the crash with different labels. It is worth noting that, except for F-measure, the other 4 indicator values on two labels are the same. This means that the 4 indicators can be used to comprehensively evaluate the overall performance of the method on the two labels. Thus, we total have 6 indicators in which the two F-measure indicator are called F-measure (InTrace) and F-measure (OutTrace). In addition, we report the results of Precision, Recall and AUC in our online supplementary materials[3].

[1]http://pitest.org/

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

[3]https://sites.google.com/view/icfr/

Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.

TABLE II: Brief Descriptions of 89 Features for Crash Instances

Feature	Description		Feature	Description
Feature Set CT and CB	Features related to the top frame CT (bottom frame CB)		Feature Set ST	Features related to the stack trace (short for ST)
CT01 (CB01)	in top (bottom) class	Number of local variables	ST01	Type of the exception in the crash
CT02 (CB02)		Number of fileds number	ST02	Number of frames of the ST
CT03 (CB03)		Function (except constructor one) number	ST03	Number of classes of the ST
CT04 (CB04)		Imported packages number	ST04	Number of functions of the ST
CT05 (CB05)		Whether the class is inherited from others	ST05	Whether an overloaded function exists in ST
CT06 (CB06)		LoC of comments	ST06	Length of the name in the top class
CT07 (CB07)	in top (bottom) function	LoC	ST07	Length of the name in the top function
CT08 (CB08)		Number of parameters	ST08	Length of the name in the bottom class
CT09 (CB09)		Number of local variable	ST09	Length of the name in the bottom function
CT10 (CB10)		Number of if-statements	ST10	Number of Java files in the project
CT11 (CB11)		Number of loops	ST11	Number of classes in the project
CT12 (CB12)		Number of for statements	Feature Set AT	Features normalized by LoC
CT13 (CB13)		Number of for-each statement	AT01 (AB01)	CT08/CT07 (CB08/CB07)
CT14 (CB14)		Number of while statements	AT02 (AB02)	CT09/CT07 (CB09/CB07)
CT15 (CB15)		Number of do-while statements	AT03 (AB03)	CT10/CT07 (CB10/CB07)
CT16 (CB16)		Number of try blocks	AT04 (AB04)	CT11/CT07 (CB11/CB07)
CT17 (CB17)		Number of catch block	AT05 (AB05)	CT12/CT07 (CB12/CB07)
CT18 (CB18)		Number of finally blocks	AT06 (AB06)	CT13/CT07 (CB13/CB07)
CT19 (CB19)		Number of assigment statements	AT07 (AB07)	CT14/CT07 (CB14/CB07)
CT20 (CB20)		Number of function calls	AT08 (AB08)	CT15/CT07 (CB15/CB07)
CT21 (CB21)		Number of return statements	AT09 (AB09)	CT16/CT07 (CB16/CB07)
CT22 (CB22)		Number of unary operators	...	...
CT23 (CB23)		Number of binary operators	AT16 (AB16)	CT23/CT07 (CB23/CB07)

C. Parameter Settings

For BDA, the factor μ in Eq. 8 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 [6]. In this work, we set 11 μ values, from 0 to 1 with a step of 0.1 and search its optimum option. In addition, we set parameter λ in Eq. 8 as 0.1. The discussion of the impacts of parameter λ values on the BDA performance, the experimental scripts, and the benchmark dataset are available in our online materials.

D. Statistic Test

To statistically analyze the significant differences among our method BDA and the baseline methods, we employ a typical non-parametric test, i.e., Friedman test (significant level at 95%) with post-hoc Nemenyi test [50]. Friedman test detects whether there exist statistically significant differences among multiple methods. When Friedman test reports a p-value lower than 0.05, the differences are considered to be significant, otherwise not. If the significant differences exist, Nemenyi test is used to find the method groups that differ with each other. Due to the drawback of traditional Nemenyi test which would generate overlapping groups, in this work, we employ its improved version in [32] to divide the methods into completely nonoverlapping groups. The improved Nemenyi test is employed in previous studies [51], [52].

V. Experimental Results

A. RQ1:How different data normalization strategies impact the performance of BDA?

Motivation: Nam et al. [9] 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 model 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 [9] 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 $\boldsymbol{x} = \{x_1, x_2, ..., x_m\}$ 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 i-th 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.

The sixth one is called **N**ormalization **A**daption **S**election (**NAS**) technique that 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 **D**ata

Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.

Characteristic **V**ector (**DCV**) which measures the similarity of the data characteristic between two projects. To calculate DCV, we first need to calculate a **D**istance **S**et (**DS**) whose elements are the Euclidean distances of all pairs of the crash instances among the data, i.e., $DS = \{d_{ij} | \forall i, j : 1 \leq i < n, i < j \leq n\}$, where d_{ij} represents the Euclidean distance between the i-th and the j-th instance. Then $DCV = \{DS\_mean, DS\_median, DS\_min, DS\_max, DS\_std, num\}$, in which the 6 entries are the mean value, the median value, the minimum value, the maximum value, the standard deviation of DS, and the number of instances, respectively. As a result, the source project and the target project individually have one DCV, represented by DCV_S and DCV_T, respectively. The similarity of the source project and the target project is measured by the similarity degree of the element in DCV_S and DCV_T using the formulation in Table III in which [e] is a element in DCV. The similarity degree ranges from "Much More" (marked as 1) to "Much Less" (marked as 7). "Much More" means the data characteristics of the two projects are very different and the element in DCV_T is larger than that in DCV_S, while "Much Less" also means the data characteristics of the two projects are very different but the element in DCV_S is larger than that in DCV_T. For example, given a element in DCV (such as the median value), if $DCV_S[median]$ and $DCV_T[median]$ satisfy the third formulation in Table III, it means that the source project and the target project are slightly different in terms of median value and $DCV_S[median]$ is smaller than $DCV_T[median]$. Here, we define a **S**imilarity **V**ector (**SV**) to represent the similarity degree of different elements in DCV. In the above example, $SV[DS\_median]$ is equal to 3.

TABLE III: Conditions for Similarity Measure

Degree	Condition
Much More (1)	$DCV\_S[e] * 1.6 < DCV\_T[e]$
More (2)	$DCV\_S[e] * 1.3 < DCV\_T[e] \leq DCV\_S[e] * 1.6$
Slightly More (3)	$DCV\_S[e] * 1.1 < DCV\_T[e] \leq DCV\_S[e] * 1.3$
Same (4)	$DCV\_S[e] * 0.9 < DCV\_T[e] \leq DCV\_S[e] * 1.1$
Slightly Less (5)	$DCV\_S[e] * 0.7 < DCV\_T[e] \leq DCV\_S[e] * 0.9$
Less (6)	$DCV\_S[e] * 0.4 < DCV\_T[e] \leq DCV\_S[e] * 0.7$
Much Less (7)	$DCV\_T[e] < DCV\_S[e] * 0.4$

After obtaining these similarity degrees, NAS technique uses the following heuristic rules to select the final normalization option which is applied to the data of the two projects before conducting our cross project ICFR model.

Rule 1: If $SV[DS\_mean]$ and $SV[DS\_std]$ are individually with degree 4, we use option N0 on both projects.

Rule 2: If $SV[DS\_min]$, $SV[DS\_max]$ and $SV[num]$ are individually either with degree 1 or 7, then we use normalization option N1 on both projects.

Rule 3: If $SV[DS\_std]$ is with degree 1 and $SV[num]$ is with degree larger than 4 or $SV[DS\_std]$ is with degree 7 and $SV[num]$ is with degree smaller than 4, then we use normalization option N3 on both projects.

Rule 4: If $SV[DS\_std]$ is with degree 1 and $SV[num]$ is with degree smaller than 4 or $SV[DS\_std]$ is with degree 7 and $SV[num]$ is with degree larger than 4, then we use normalization option N4 on both projects.

Rule 5: If none of the above 4 rules are satisfied, then we use normalization option N2 on both projects.

Results: Figure 3 shows the box-plots of the 6 indicator values for our cross project ICFR model BDA with the 6 data normalization techniques. From the figure, we observe that BDA with normalization techniques N0 and N1 achieve the worst performance in terms of F-measure (InTrace), G-measure, g-mean, Balance, and MCC on nearly all cross project pairs, but gets the best median F-measure (OutTrace). It means that BDA with these two normalization techniques identify the residences of all the crashing faults as label 'OutTrace'. Thus, they make no sense for ICFR task even although they obtain the best F-measure (OutTrace). For N2, N3, N4, NAS options, BDA with N2 achieves the higher median F-measure (InTrace), G-measure, g-mean, Balance, and MCC values than BDA with other 3 options, while the median F-measure (OutTrace) by BDA with N2 is only lower than that with N3. Thus, overall, normalization technique N2 is more suitable than the others. This conclusion is somewhat inconsistent with that in [9] which suggested that normalization technique NAS is the most suitable choice. In this work, we normalize the cross project data with the N2 option before conducting the BDA method.

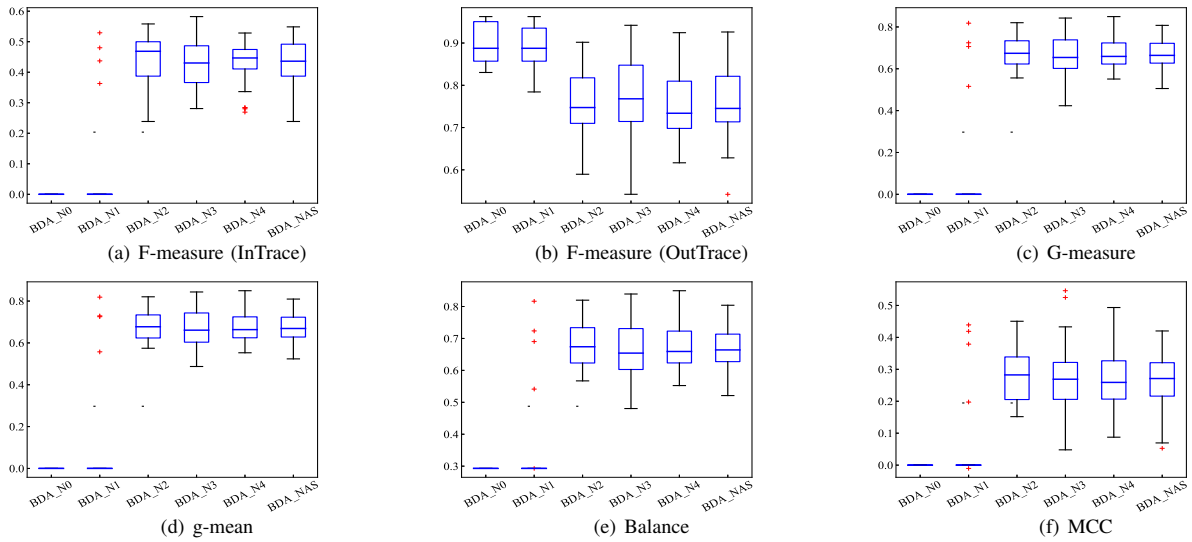
B. RQ2:Does BDA perform better than the variant methods of its downgraded versions?

Motivation: As BDA incorporates both the marginal and conditional distributions during the feature embedding process and assigns different weights to the two distributions for distinct cross project pairs, this question is designed to study whether BDA is superior to its variant methods which only consider one of the distribution or joint the two distributions with equal weights.

Method: To answer this question, we employ a basic method (called "None") and 3 downgraded versions of BDA for comparison, including TCA [48], CDT, and JDA [47]. "None" does not involve any feature transformation. TCA only focuses on the margin distribution. JDA considers two distributions with the same weight. CDT is proposed by us which only concerns about the conditional distribution. In this study, we also combine N2 option with all baseline methods for fair comparison.

Results: The detailed results for the baseline methods are available in our online supplementary materials. Figure 4 depicts the box-plots of the 6 indicators for the 5 methods. Figure 5 visualizes the statistical results (i.e., the CD diagrams) of post-hoc test by Nemenyi test after Friedman test on each indicator. The term CD in the upper left corner of each subfigure denotes the critical difference of the Nemenyi test, and the upper right corner shows the p value of the Friedman test. From Figures 4 and 5, we have the following findings:

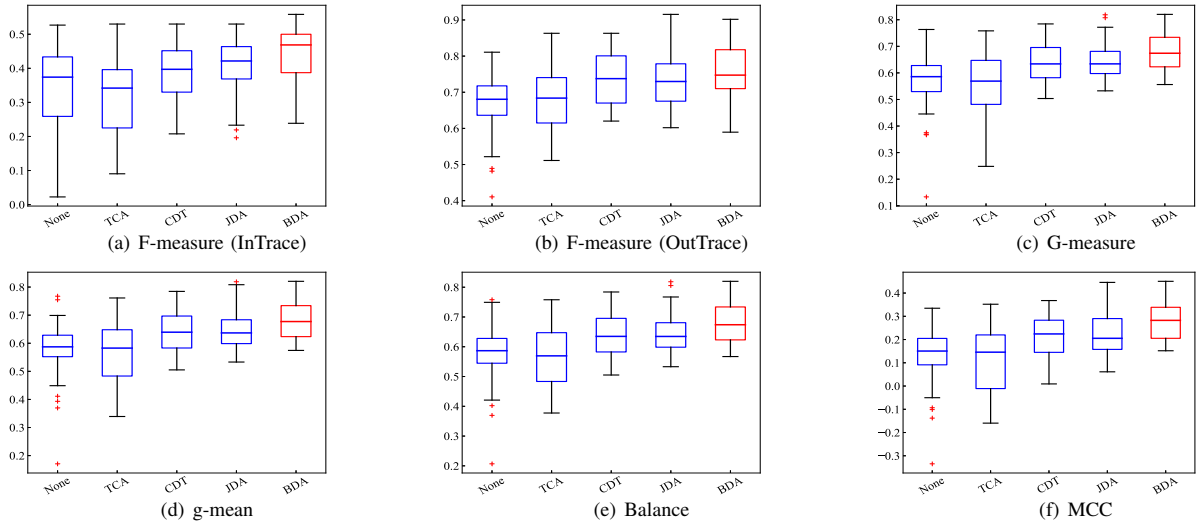
Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.



(a) F-measure (InTrace) (b) F-measure (OutTrace) (c) G-measure

(d) g-mean (e) Balance (f) MCC

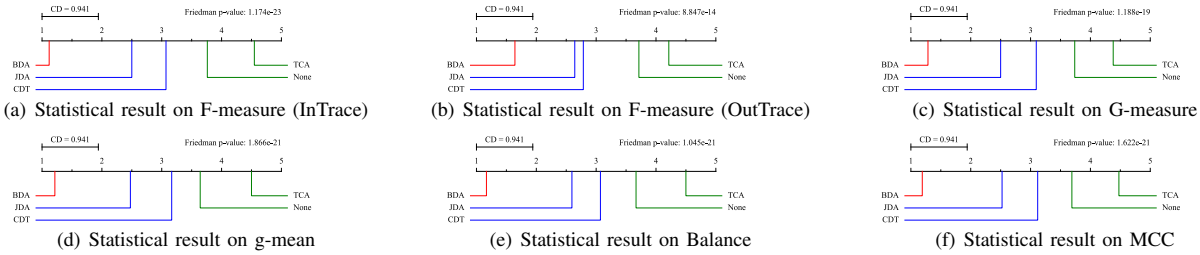
Fig. 3: Box-plots of the 6 indicator values for BDA with 6 data normalization techniques.



(a) F-measure (InTrace) (b) F-measure (OutTrace) (c) G-measure

(d) g-mean (e) Balance (f) MCC

Fig. 4: Box-plots of the 6 indicator values for for BDA, its 3 variant methods and a most basic method.



(a) Statistical result on F-measure (InTrace) (b) Statistical result on F-measure (OutTrace) (c) Statistical result on G-measure

(d) Statistical result on g-mean (e) Statistical result on Balance (f) Statistical result on MCC

Fig. 5: Comparison of average ranks of BDA and the 4 baselines methods with Nemenyi test in terms of the 6 indicators. Methods that are significantly different are distinguished with different color.

Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.

First, From Figure 4, it can be seen that BDA achieves the best median performance compared with the 4 baseline methods in terms of all indicators. More specifically, the superiority of median performance by BDA is obviously greater than those by None and TCA in all indicators. Whereas the superiority of median F-measure (OutTrace) by BDA is slight when compared with that by CDT and JDT.

Second, the basic method None is outperformed by CDT, JDT, and BDA, while is only slightly better than TCA. This means that using transfer learning methods is necessary to improve the cross project ICFR performance, but only considering the margin distribution has no positive effects. When comparing with CDT and JDT, they achieve nearly the same median indicator values in terms of 5 out of 6 indicators except for F-measure (InTrace). It means that using the same weight to combine the two distributions can not achieve better ICFR performance than only considering the conditional distribution.

Third, the p values in the sub-figures of Figure 5 show that there exist statistically significant differences among the 5 methods in terms of all indicators. The CD diagrams illustrate that our cross project ICFR model BDA performs significantly better than the 4 baseline methods in terms of all indicators. In addition, the CD diagrams also show that the average rank of JDT is higher than that of CDT in terms of all indicators, but with the same blue color. It means that considering both margin and conditional distribution is helpful to achieve better ICFR performance, but the differences are not significant compared with considering only the conditional distribution.

C. RQ3:Is BDA superior to the other cross project models?

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

Method: We choose total 9 cross project models that are originally designed for software fault prediction task as our baseline methods, including 4 instance filtering based models: NN-Filter [22], Peter-Filter [23], Yu-Filter [24], and HISNN [53]; 2 transfer learning based models: TCA+ [9] and TNB [26]; one feature selection based model: FeSCH [54]; and 2 classifier combination based models: CODEP [28] and ASCI [29].

Results: Figure 6 depicts the box-plots of the 6 indicators for the whole 10 models. Figure 7 visualizes the corresponding statistical results on each indicator. From Figure 6 and 7, we have the following observations:

First, Figure 6 depicts that BDA achieves the best median performance compared with the 9 baseline methods in terms of 5 indicators expect for F-measure (OutTrace). More specifically, the median performance by BDA is obviously superior to that by the 9 methods in the 5 indicators. Whereas BDA is outperformed by two classifier combination based cross project models in terms of F-measure (OutTrace).

Second, among the 4 instance filtering based cross project models, Yu-Filter obtains the highest median performance in terms of 5 indicators and the similar median F-measure (InTrace) to NN-Filter and Peter-Filter, while HISNN performs the worst in all indicators. Among the 2 transfer learning based cross project models, TCA+ and TNB achieve the similar median performance on 2 F-measure indicators while TCA+ achieves much better median performance on other 4 comprehensive indicators. Among the 2 classification combination based cross project models, they obtain the similar median performance on all 6 indicators.

Third, the p values in Figure 7 show that the 10 methods have significant differences in terms of all indicators. The CD diagrams illustrate that our method BDA performs significant better than the 9 baseline methods in terms of 5 indicators expect for F-measure (OutTrace), whereas BDA has no significant differences compared with CODEP and ASCI in terms of F-measure (OutTrace).

VI. DIACUSSION

In this section, we give some implications derived from our experimental results.

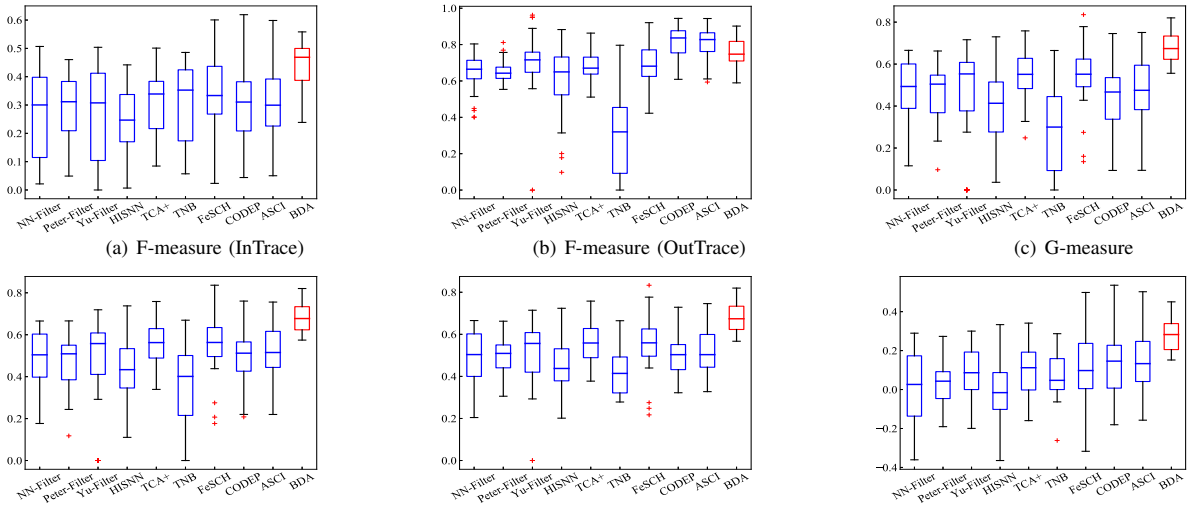
From the RQ1, we observe that applying z-score technique to the cross project data individually (i.e., the N2 option) is sufficient to achieve better ICFR performance. Thus, we do not need to utilize the intricate rules to select the optimal normalization technique (i.e., the NAS option) for specific cross project pair over our benchmark dataset. The results of RQ2 show that considering both distributions in transfer learning with distinct weights is more effective to promote the cross project ICFR performance than the methods using equal weights or only considering one distribution. This denotes that 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 RQ3, we observe that BDA shows obvious superiority towards the typical cross project models designed for other learning tasks overall. This means that the BDA is more suitable for our cross project ICFR task. But there is still improvement room for F-measure (OutTrace) performance of BDA since it is not always the best.

VII. THREATS TO VALIDITY

A. 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. In addition, the dataset is collected via imitating the crashes caused by the seeded faults using program mutation operations. Since such simulative crashes could not fully reflect the real ones in practical scenarios, thus, experiments on real-world crashes are needed to extend the generalization of our results.

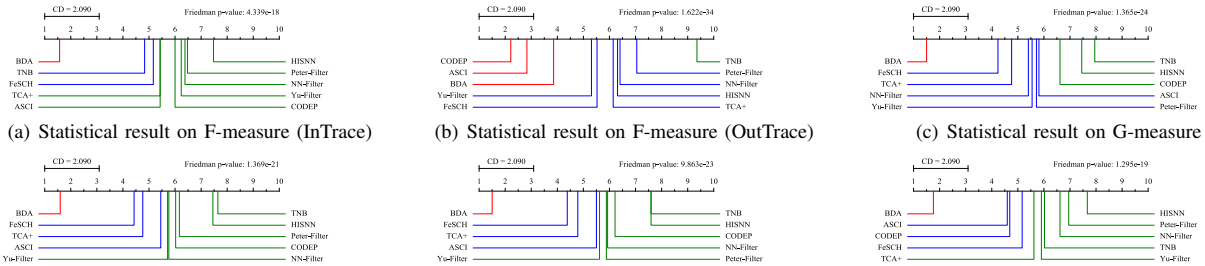
Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.



(a) F-measure (InTrace) (b) F-measure (OutTrace) (c) G-measure

(d) g-mean (e) Balance (f) MCC

Fig. 6: Box-plots of the 6 indicator values for BDA nd 9 baseline cross project models.



(a) Statistical result on F-measure (InTrace) (b) Statistical result on F-measure (OutTrace) (c) Statistical result on G-measure

(d) Statistical result on g-mean (e) Statistical result on Balance (f) Statistical result on MCC

Fig. 7: Comparison of average ranks of 10 cross project models with Nemenyi's post hoc test in terms of the 6 indicators.

B. 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.

C. 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.

VIII. Conclusion

To resolve the dilemma of label shortage for ICFR task, we propose a BDA based cross project ICRF model by leveraging the labeled data of other projects to carry out the learning task on the unlabeled target project data. BDA reduces the marginal and conditional distribution discrepancies between the cross project data simultaneously by assigning them with different weights based on the data distribution similarity of the two projects. Experiments on 7 open-source Java projects show that our cross project model BDA achieves better ICFR performance than the baseline methods on 6 indicators overall.

In future, we plan to explore a feasible way to automatically specify the weights for the two distributions and investigate the impacts of the class imbalance on the performance of BDA.

Acknowledgment

This work was supported by National Key Research and Development Program of China (2018YFC1604000), the grands of the National Natural Science Foundation of China (Nos.61572374, 61602258, U163620068), Natural Science Foundation of Heilongjiang Province (No.LH2019F008), the Open Fund of Key Laboratory of Network Assessment Technology from Chinese Academy of Sciences, Hong Kong GRC Project (Nos.PolyU 152223/17E, PolyU 152239/18E), the General Research Fund of the Research Grants Council of Hong Kong (No.11208017), the research funds of City University of Hong Kong (Nos.9678149, 7005028), and the Research Support Fund by Intel (No.9220097).

Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *Transactions on Software Engineering (TSE)*, vol. 42, no. 8, pp. 707–740, 2016.

[2] Y. Gu, J. Xuan, H. Zhang, L. Zhang, Q. Fan, X. Xie, and T. Qian, "Does the fault reside in a stack trace? assisting crash localization by predicting crashing fault residence," *Journal of Systems and Software (JSS)*, vol. 148, pp. 88–104, 2019.

[3] Y. Kultur, B. Turhan, and A. B. Bener, "Enna: software effort estimation using ensemble of neural networks with associative memory," in *Proceedings of the 16th SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 2008, pp. 330–338.

[4] E. Kocaguneli, B. Cukic, T. Menzies, and H. Lu, "Building a second opinion: learning cross-company data," in *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*, 2013, p. 12.

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

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

[7] F. Zhang, I. Keivanloo, and Y. Zou, "Data transformation in cross-project defect prediction," *Empirical Software Engineering (ESE)*, vol. 22, no. 6, pp. 3186–3218, 2017.

[8] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," *Information and Software Technology (IST)*, pp. 125–136, 2018.

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

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

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

[12] T. Dhaliwal, F. Khomh, and Y. Zou, "Classifying field crash reports for fixing bugs: A case study of mozilla firefox," in *Proceedings of the 27th International Conference on Software Maintenance (ICSM)*, 2011, pp. 333–342.

[13] N. Chen and S. Kim, "Star: Stack trace based automatic crash reproduction via symbolic execution," *Transactions on Software Engineering (TSE)*, vol. 41, no. 2, pp. 198–220, 2015.

[14] M. Nayrolles, A. Hamou-Lhadj, S. Tahar, and A. Larsson, "A bug reproduction approach based on directed model checking and crash traces," *Journal of Software: Evolution and Process*, vol. 29, no. 3, p. e1789, 2017.

[15] ——, "Jcharming: A bug reproduction approach using crash traces and directed model checking," in *Proceedings of the 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2015, pp. 101–110.

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

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

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

[19] A. Schroter, A. Schröter, N. Bettenburg, and R. Premraj, "Do stack traces help developers fix bugs?" in *Proceedings of the 7th Working Conference on Mining Software Repositories (MSR)*, 2010, pp. 118–121.

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

[21] S. Jiang, W. Li, H. Li, Y. Zhang, H. Zhang, and Y. Liu, "Fault localization for null pointer exception based on stack trace and program slicing," in *Proceedings of the 12th International Conference on Quality Software (ICQS)*, 2012, pp. 9–12.

[22] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering (ESE)*, vol. 14, no. 5, pp. 540–578, 2009.

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

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

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

[26] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology (IST)*, vol. 54, no. 3, pp. 248–256, 2012.

[27] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Negative samples reduction in cross-company software defects prediction," *Information and Software Technology (IST)*, vol. 62, pp. 67–77, 2015.

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

[29] D. Di Nucci, F. Palomba, and A. De Lucia, "Evaluating the adaptive selection of classifiers for cross-project bug prediction," 2018.

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

[31] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, "Building an ensemble for software defect prediction based on diversity selection," in *Proceedings of the 10th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2016, p. 46.

[32] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *Transactions on Software Engineering (TSE)*, 2017.

[33] Y. Zhou, Y. Yang, H. Lu, L. Chen, Y. Li, Y. Zhao, J. Qian, and B. Xu, "How far we have progressed in the journey? an examination of cross-project defect prediction," *Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 1, p. 1, 2018.

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

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

[36] L. L. Minku and S. Hou, "Clustering dycom: An online cross-company software effort estimation study," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2017, pp. 12–21.

[37] E. Mendes and B. Kitchenham, "Further comparison of cross-company and within-company effort estimation models for web applications," in *Proceedings of the 10th International Symposium on Software Metrics*, 2004, pp. 348–357.

[38] B. A. Kitchenham and E. Mendes, "A comparison of cross-company and within-company effort estimation models for web applications," in *Proceedings of the 8th International Conference on Empirical Assessment in Software Engineering (EASE)*, 2004, pp. 47–55.

[39] E. Mendes, S. Di Martino, F. Ferrucci, and C. Gravino, "Effort estimation: how valuable is it for a web company to use a cross-company data set, compared to using its own single-company data set?" in *Proceedings of the 16th International Conference on World Wide Web (WWW)*, 2007, pp. 963–972.

Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.

[40] R. Malhotra and A. J. Bansal, "Cross project change prediction using open source projects," in *Proceedings of International Conference on Advances in Computing, Communications and Informatics*, 2014, pp. 201–207.

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

[42] A. Bansal and S. Jajoria, "Cross-project change prediction using meta-heuristic techniques," *International Journal of Applied Metaheuristic Computing (IJAMC)*, vol. 10, no. 1, pp. 43–61, 2019.

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

[44] S. Lal, N. Sardana, and A. Sureka, "Three-level learning for improving cross-project logging prediction for if-blocks," *Journal of King Saud University-Computer and Information Sciences*, pp. 1–16, 2017.

[45] ——, "Eclogger: Cross-project catch-block logging prediction using ensemble of classifiers," *e-Informatica Software Engineering Journal*, vol. 11, no. 1, 2017.

[46] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, 2015, pp. 415–425.

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

[48] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *Transactions on Neural Networks (TNN)*, vol. 22, no. 2, pp. 199–210, 2011.

[49] Z. Lin, M. Chen, and Y. Ma, "The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices," *arXiv preprint arXiv:1009.5055*, 2010.

[50] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, no. Jan, pp. 1–30, 2006.

[51] Z. Xu, S. Li, X. Luo, J. Liu, T. Zhang, Y. Tang, J. Xu, P. Yuan, and J. Keung, "Tstss: A two-stage training subset selection framework for cross version defect prediction," *Journal of Systems and Software (JSS)*, vol. 154, pp. 59–78, 2019.

[52] Z. Xu, S. Pang, T. Zhang, X. Luo, J. Liu, Y. Tang, X. Yu, and L. Xue, "Cross project defect prediction via balanced distribution adaptation based transfer learning," *Journal of Computer Science and Technology (JCST)*, accepted to appear, 2019.

[53] D. Ryu, J.-I. Jang, and J. Baik, "A hybrid instance selection using nearest-neighbor for cross-project defect prediction," *Journal of Computer Science and Technology (JCST)*, vol. 30, no. 5, pp. 969–980, 2015.

[54] C. Ni, W.-S. Liu, X. Chen, Q. Gu, D.-X. Chen, and Q.-G. Huang, "A cluster based feature selection method for cross-project software defect prediction," *Journal of Computer Science and Technology (JCST)*, vol. 32, no. 6, pp. 1090–1107, 2017.

Authorized licensed use limited to: Wuhan University. Downloaded on October 08,2021 at 02:00:45 UTC from IEEE Xplore. Restrictions apply.