




Cross-project bug type prediction based on transfer learning

Xiaoting Du¹  · Zenghui Zhou¹ · Beibei Yin¹ · Guanping Xiao¹

Published online: 16 September 2019
© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract

The prediction of bug types provides useful insights into the software maintenance process. It can improve the efficiency of software testing and help developers adopt corresponding strategies to fix bugs before releasing software projects. Typically, the prediction tasks are performed through machine learning classifiers, which rely heavily on labeled data. However, for a software project that has insufficient labeled data, it is difficult to train the classification model for predicting bug types. Although labeled data of other projects can be used as training data, the results of the cross-project prediction are often poor. To solve this problem, this paper proposes a cross-project bug type prediction framework based on transfer learning. Transfer learning breaks the assumption of traditional machine learning methods that the training set and the test set should follow the same distribution. Our experiments show that the results of cross-project bug type prediction have significant improvement by adopting transfer learning. In addition, we have studied the factors that influence the prediction results, including different pairs of source and target projects, and the number of bug reports in the source project.

Keywords Bug prediction · Cross-project · Bug report · Transfer learning

1 Introduction

Bugs are inevitably existed in software projects, even though these projects are developed by experienced developers using powerful development platforms. Predicting bug types is

✉ Xiaoting Du
xiaoting_2015@buaa.edu.cn

Zenghui Zhou
zhouzenghui@buaa.edu.cn

Beibei Yin
yinbeibei@buaa.edu.cn

Guanping Xiao
gpxiao@buaa.edu.cn

¹ School of Automation Science and Electrical Engineering, Beihang University, Beijing, China

very beneficial for helping developers analyze bug characteristics and conduct corresponding solutions, such as developing automatic detection tools or fault tolerance techniques to eliminate the same/similar bugs in the future releases (Trivedi et al. 2011). Several studies use machine learning classifiers to perform bug type prediction tasks, whose models are trained using software metrics calculated from source files (Qin et al. 2018) or text information in bug reports (Zhou et al. 2016). For example, Wen et al. (2016) proposed a two-step automated approach that integrates natural language processing, information retrieval, and machine learning to predict configuration bugs. Javed et al. (2012) used Multinomial Naive Bayes classifier to classify bugs into different labels. Du et al. (2017) utilized seven state-of-the-art machine learning classifiers to automatically classify bugs based on fault triggering conditions.

When predicting bug types, training data and test data fed into machine learning classifiers are usually from the same project. However, several reasons may lead to the lack of labeled data of a software project. For example, a newly released project may have a few amounts of bug data; it is not cost-effective to collect or label bug data; bug data is not available to be accessed. As a result, it would not have enough labeled data to perform within-project bug type prediction tasks. Intuitively, we can use labeled data from other projects that have similar types of bugs to conduct the cross-project bug type prediction (Qin et al. 2018). In this circumstance, training data and test data that come from different projects cannot satisfy the assumption of traditional machine learning methods, i.e., the data distributions of training and test sets should be identical (Weiss et al. 2016). Consequently, compared with the within-project bug type prediction, the cross-project bug type prediction would have a lower performance. Therefore, how to improve the performance of cross-project bug type prediction is a crucial issue of this paper.

Transfer learning breaks the assumption of traditional machine learning methods. When the amount of training data is insufficient in one domain, it can be used to improve the learner by transferring information from a related domain (Pan and Yang 2010). Due to this advantage, transfer learning has many applications, such as text sentiment classification and multilingual text classification (Wang and Mahadevan 2011; Zhou et al. 2014). In software engineering research, transfer learning has been used in cross-project bug prediction tasks (He et al. 2012; Jing et al. 2015), which predicts bugs in source code based on software metrics. Different from existing work, we aim to predict the types of bugs based on text information contained in bug reports. Bug reports from different projects have several common features. For example, the “summary” section of bug reports is described in the form of natural language and is used to summarize issues encountered by developers/users during the usage of a software system. Thus, it is possible to use transfer learning to migrate useful information from another project to predict bug types of a project based on the text information in bug reports.

In this paper, we propose a cross-project bug type prediction framework based on transfer learning by using text information contained in the “summary” section of bug reports. The transfer learning method used in this paper is TrAdaBoost (Dai Wenyuan et al. 2007b). We have studied the effectiveness of the proposed framework and the influencing factors of the transferring performance. This paper mainly focuses on answering the following three research questions.

RQ1: How to use transfer learning to perform cross-project bug type prediction?

RQ2: How do different pairs of source and target projects affect the bug type prediction results?

RQ3: How does the number of bug reports from the source project impact the cross-project bug type prediction results?

This paper makes the following main contributions.

1. A cross-project bug type prediction framework based on transfer learning is proposed and the impact of transfer learning on cross-project bug type prediction is analyzed.
2. The transferring performance between six different pairs of source and target projects was analyzed, including MySQL→HTTPD, AXIS→HTTPD, MySQL→AXIS, HTTPD→AXIS, HTTPD→MySQL, and AXIS→MySQL.
3. The impact of the number of bug reports from a source project is analyzed by proportionally increasing the number of the source project’s bug reports.

The remainder of this paper is organized as follows. The approach is introduced in Section 2. Section 3 presents the experiment setups. Section 4 shows the experiments and analyzes the experimental results. Section 5 discusses the threats to validity. Section 6 introduces the related work. Finally, conclusion and discussion are presented in Section 7.

2 Approach

2.1 Framework

The framework consists of five steps, i.e., data collection, data pre-processing, bug report representation, transfer learning, and bug type prediction, as shown in Fig. 1.

Step 1. Data collection: The first step of the framework is data collection. Bug reports used in this paper come from four projects, i.e., Linux, MySQL, HTTPD, and AXIS. For each project, bug reports are classified into Bohrbugs (BOH) and Mandelbugs (MAN) (Grottke and Trivedi 2005). Bohrbug is a kind of bug that is easy to isolate, and whose failure manifestation is consistent under a well-defined set of triggering conditions. In contrast, Mandelbug is a kind of bug whose activation and/or error propagation is

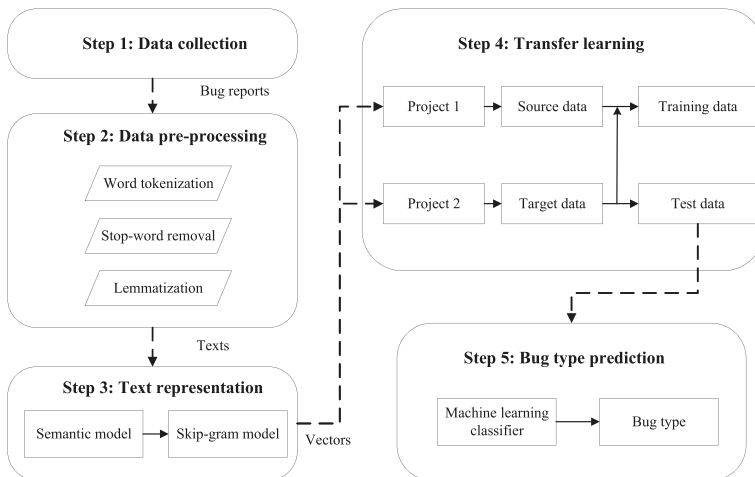


Fig. 1 A transfer-learning-based cross-project bug type prediction framework

complex, and they are difficult to be reproduced by traditional techniques. In this study, the labeled bug reports of the Linux project are from Xiao et al. (2017), while the labeled bug reports of the MySQL, HTTPD, and AXIS projects are from Cotroneo et al. (2013). The “summary” section of bug reports is taken as the feature of our framework.

Step 2. Data pre-processing: The second step is data pre-processing. After the data collection, it is necessary to pre-process bug reports because they may have several meaningless characters in the “summary” section. The pre-processing step in this paper includes word tokenization, stop-word removal, and lemmatization. These operations are elaborated in Section 2.2.

Step 3. Bug report representation: Bug reports are represented by a semantic model trained using word2vec, a word embedding tool proposed by Mikolov et al. (2013b). For the training of the semantic model, 134277 bug reports were downloaded from bug tracking systems (e.g., Bugzilla and JIRA). The text information (e.g., summary and description) in bug reports were extracted as raw corpora for training the semantic model. In the semantic model, each word from the bug reports can be represented as a word vector. The distance between two vectors can be used to measure the semantic relationship between two words. A vector representation of each bug report is obtained by averaging the word vectors of words in its “summary” section. In Section 2.3, we present details of the step.

Step 4. Transfer learning: The project (i.e., target project) that is needed to perform bug type prediction may not have enough labeled bug reports for training the classification model, while other projects (i.e., source projects) may have a large amount of label data. Transfer learning can capture useful information from the source project as additional training data for the target project’s prediction. In this study, we use TrAdaBoost, a classic transfer learning method proposed by Dai Wenyuan et al. (2007b). The detailed procedure of using TrAdaBoost is shown in Section 2.4.

Step 5. Bug type prediction: Machine learning classifiers are used to predict the type of bugs in this step. To compare the performance of different classifiers, three machine learning classifiers are used, including Gradient Boosting Classifier (GBC), SGDClassifier (SGD), and AdaBoostClassifier (ABC). The evaluation metrics used in this paper are precision, recall, and F -measure.

2.2 Data pre-processing

The text contained in the “summary” section of each bug report is pre-processed in three steps, i.e., word tokenization, stop-word removal, and lemmatization. We use built-in functions provided by the NLTK toolkit to perform these steps (Loper and Bird 2002).

- 1. Word tokenization:** It is a basic step of text pre-processing, which divides the text into a stream of words or meaningful elements. In this step, numbers and punctuations that appear in the text are removed. The other non-alphabetic characters (e.g., “#”, “*”, and “&”) are replaced with spaces. After these two operations, the remaining words in the “summary” section of bug reports are tokenized into a word list. For example, given a report’s summary “Kernel hangs when APIC/ACPI enabled,” after the word tokenization, we obtain [“kernel,” “hangs,” “when,” “apic,” “acpi,” “enabled”].
- 2. Stop-word removal:** In information retrieval and text mining, many of the most frequently used words in English are useless, which are called stop words. These words are frequently used words but contain no information (e.g., “and,” “the,” “to”). Removing stop words in the “summary” usually would not affect the understanding of bugs. For

example, for a summary in a bug report “Changing the CPU frequency using CPUFREQ hangs the kernel,” the word “the” is removed in this step. It is an important step in text pre-processing to exclude unimportant information by removing stop words (Silva and Ribeiro 2003).

3. **Lemmatization:** We perform lemmatization to find the canonical form of a word. After lemmatization, we can obtain a meaningful and complete word, which is generally a valid word in the dictionary. For example, the word “aliases” in the bug summary (e.g., “broken module aliases in ieeec1394 drivers”) is transformed into its standard form “alias” after lemmatization. Lemmatization is often used for text mining and natural language processing fields (Plisson et al. 2004).

2.3 Bug report representation

We use word2vec to train a semantic model to represent bug reports. Two models are available in word2vec, including continuous bag-of-words model and continuous skip-gram model (Mikolov et al. 2013a, b). In word2vec, deep neural networks are used to learn the semantic information contained in the context of corpora to generate low-dimensional word vector representations, i.e., “word embedding.” For the continuous bag-of-words model, it predicts the current word based on its context, while the skip-gram model predicts the surrounding words given a current word.

In our study, we use the skip-gram model, which has been shown to work well in solving software engineering tasks (Yang et al. 2016). Given a current word w_t , we denote the set of the context of w_t as $Context_{w_t}$. For example, the summary of the Linux bug ID-4707 is “memory leak somewhere in toshiba_acpi.” Figure 2 denotes the training process of the skip-gram model. The current word $w_t = somewhere$ is mapped to its word vector v_{w_t} . The skip-gram model predicts the word vectors of the current word’s left and right C surrounding words (e.g., $C = 2$). $Context_{w_t} = \{memory, leak, toshiba, acpi\}$ are the context words of $w_t = somewhere$. The training process optimizes the word

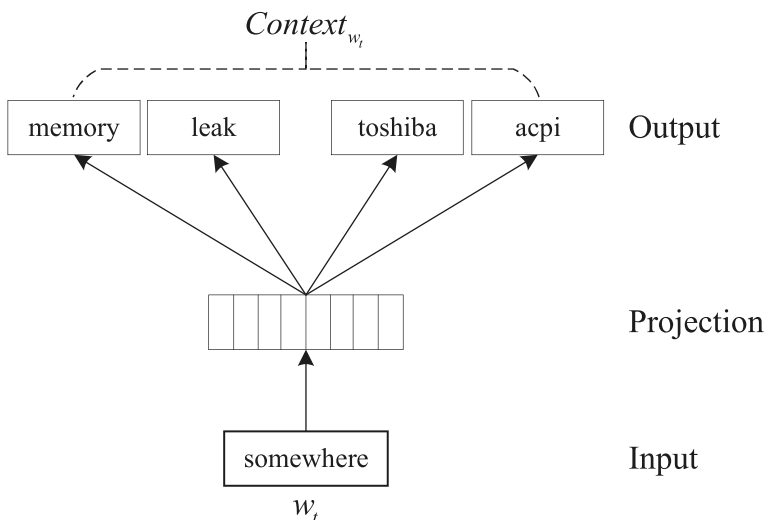


Fig. 2 Example of Skip-gram

embedding v_{w_t} and the neural network model parameters aiming to maximize the objective function f :

$$f = \frac{1}{T} \sum_{t=1}^T \sum_{w_c \in \text{Context}_{w_t}} \log p(w_c | w_t) \quad (1)$$

where w_c is the word in the context of w_t . T is the length of the word sequence. The probability $p(w_c | w_t)$ is formulated using a soft-max function:

$$p(w_c | w_t) = \frac{\exp(v_{w_c}^T \cdot v_{w_t})}{\sum_{j=1}^W \exp(v_{w_j}^T \cdot v_{w_t})} \quad (2)$$

where v_{w_c} is the word vector of word w_c , and W is the length of word vocabulary.

The vector of a bug report is obtained by averaging the vectors of all the words in its “summary” section. Thus, each bug report can be represented as a vector, which represents the characteristic of the bug report.

2.4 Transfer learning

The transfer learning method used in this study is TrAdaBoost. According to different situations of tasks between the source and target domain, transfer learning has three types (Pan and Yang 2010), including inductive transfer learning, transductive transfer learning, and unsupervised transfer learning. Furthermore, each type can be summarized into four cases based on “What to transfer,” i.e., instance-based transfer learning, feature-representation-transfer approach, parameter-transfer approach, and relational-knowledge-transfer problem. TrAdaBoost, one of the typical methods, belongs to instance-based transfer approach of the inductive transfer learning (Dai Wenyuan et al. 2007b). The advantages of using TrAdaBoost are as follows.

First, the characteristics of bug data in this paper satisfy the requirements of TrAdaBoost, which assumes that the data of source project and target project have the same set of features and labels, but the distributions are different. The feature representations of bug reports in the source project and target project are denoted by word vectors, which means the data features of the source project and target project are the same. In addition, the prediction labels for the source project and target project are the same, i.e., Bohrbug and Mandelbug. However, the source data and target data come from different projects, which means that data distributions are different (Qin et al. 2018).

Besides, the cross-project bug type prediction well meets the application scenario of TrAdaBoost. The goal of cross-project bug type prediction is to predict bug types of target project by using little data of the target project and a large amount of data from the source project. Due to the difference in distributions between the source and the target domains, some of the data in the source project may be useful for the target project’s bug type prediction, while some of them may be harmful. TrAdaBoost attempts to adjust the weight of the source data through experimental iterations to reduce the impact of “bad” source data and to leverage the impact of “good” source data on improving the target project’s prediction performance.

Figure 3 shows the training procedure of TrAdaBoost algorithm. Assuming Linux is the source project and MySQL is the target project, we use X_S and X_T to represent the

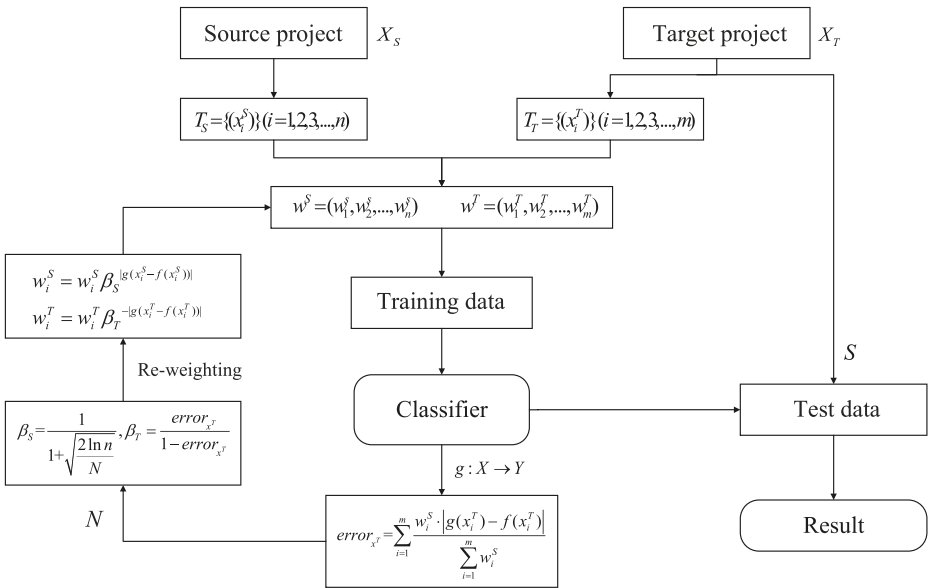


Fig. 3 Procedure of TrAdaBoost

sample space of source project and target project, and $Y = \{0, 1\}$ to indicate the set of bug types. In this study, $y = 1$ denotes a Bohrbug and $y = 0$ represents a Mandelbug. The boolean function g is the prediction function from the sample space X ($X = X_T \cup X_S$) to the category label Y , i.e., the classification algorithm of bug types. Training data $T \subseteq \{X \times Y\}$ contains two parts of labeled data $T_S = \{(x_i^S, g(x_i^S))\}$ and $T_T = \{(x_i^T, g(x_i^T))\}$, where $x_i^S \in X_S$ ($i = 1, \dots, n$) and $x_i^T \in X_T$ ($i = 1, \dots, m$). The number of labeled bug reports in Linux and MySQL is denoted by n and m , respectively. Test data, i.e., unlabeled data in MySQL, is $S = \{(x_i^S)\}$, where $x_i^S \in X_S$ ($i = 1, \dots, k$). k is the number of unlabeled bugs.

For each iteration (total number is N), if the training sample of Linux is predicted wrongly, the sample is likely to conflict with training data of MySQL. Then, the misclassified Linux training sample will be assigned a smaller weight w_i^S by multiplying the weight β^S ($\beta^S \in (0, 1]$), which means less impact on the learning process. After several iterations, the Linux training samples that match the MySQL data will have more weight. In contrast, Linux training samples that differ from MySQL’s data will have less weight. Samples with larger training weights can help train better classifiers.

3 Experiment setups

3.1 Datasets

Table 1 shows the datasets of four projects in our experiments, including Linux, MySQL, HTTPD, and AXIS. The dataset of Linux is from Xiao et al. (2017), containing 2444 BOHs and 1590 MANs, while the datasets of MySQL, HTTPD, and AXIS are from Cotroneo et al. (2013). Since the number of bug reports in Linux is about 10 times more than that of other

Table 1 Bug report dataset

Project	No. of BOH	No. of MAN	No. of reports
Linux	2444	1590	4034
MySQL	125	84	209
HTTPD	116	25	141
AXIS	184	15	199

projects, Linux is mainly used as a source project. In this study, 20% of bug reports of the target project and all the bug reports from the source project are used as training data, while the remaining 80% of the target project data are used as test data.

3.2 Classifiers

To compare the performance of different classifiers, three classifiers, i.e., Gradient Boosting Classifier (GBC), SGDClassifier (SGD), and AdaBoostClassifier (ABC), are used in the proposed framework. The classifiers are implemented by scikit-learn, a python module that integrates several state-of-the-art machine learning algorithms (Pedregosa et al. 2011). In the following, we briefly introduce these classifiers.

- 1. Gradient Boosting Classifier (GBC):** Gradient Tree Boosting is an ensemble algorithm. Its motivation is to combine several weak models to produce a powerful ensemble (Friedman 2001). As any other boosting methods, it constructs models in a phased manner and generalizes them by allowing the optimization of arbitrary differential loss functions.
- 2. SGDClassifier (SGD):** Stochastic gradient descent is a very simple and effective method for fitting linear models (e.g., support vector machine and logistic regression; Zadrozny and Elkan 2002), The model fitted here is the support vector machine, which is very useful when dealing with large data and large feature quantities and has been applied to solve large-scale and sparse machine learning problems in the fields of text classification and natural language processing.
- 3. AdaBoostClassifier (ABC):** AdaBoost is a meta-estimator, which first fits the classifier on the original dataset and then fits other copies of the classifier on the same dataset. Since the weights of the instances misclassified are adjusted, the subsequent classifiers focus more on difficult cases (Freund and Schapire 1997).

3.3 Evaluation metrics

In this study, we use precision, recall, and F -measure to evaluate the performance of classifiers. These metrics are calculated based on a confusion matrix which records correctly and incorrectly predicted instances for each class (Sokolova et al. 2006). Table 2 shows a binary classification confusion matrix of BOH/MAN. The column of the matrix records the predicted label of the bug reports and the row of the matrix provides the actual label of the bug reports. A cell in a matrix represents the number of bug reports for a particular predicted label and a specific actual label. In a binary classification confusion matrix, four cells exist, i.e., a bug report is classified as a BOH and it is indeed a BOH (true positive, TP); a bug report is classified as a BOH but it is actually a MAN (false positive, FP); a bug report is actually a BOH but it is classified as a MAN (false negative, FN); a bug report is classified

Table 2 Confusion matrix for binary classification

	Predicted: BOH	Predicted: MAN
Actual: BOH	True positive (TP)	False negative (FN)
Actual: MAN	False positive (FP)	True negative (TN)

as a MAN and it is indeed a MAN (true negative, TN). The calculation of the selected metrics is presented as follows.

- 1. Precision:** It estimates the proportion of a class predicted correctly over all instances which are classified into this class. The metric can assess the predictive power of the algorithm. The definition is as follows:

$$\text{precision} = \frac{TP}{TP + FP} \quad (3)$$

- 2. Recall:** Recall measures the portion of correctly predicted members over all actual class members. Recall is calculated as:

$$\text{recall} = \frac{TP}{TP + FN} \quad (4)$$

- 3. F -Measure:** F -Measure is a composite measure that combines both precision and recall, which evaluates if an increase in precision outweighs a reduction in recall. If precision and recall are equally important, F -measure can be used. The formula of F -measure is:

$$F - \text{measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (5)$$

4 Results and analysis

4.1 The impact of transfer learning on cross-project bug type prediction

In this part, we analyze the effect of transfer learning on cross-project bug type prediction. The source project used in this part is Linux, while the target projects are MySQL, HTTPD, and AXIS. Figure 4 shows the F -measure of cross-project bug type prediction with transfer learning and without transfer learning. For all the three target projects, the F -measure of the prediction with transfer learning outperforms that without transfer learning. In addition, the classifiers have a different impact on the prediction results, as shown in Table 3.

For the prediction with transfer learning, GBC classifier has the best performance on all the three target projects. For HTTPD, MySQL, and AXIS projects, the prediction results, i.e., (precision, recall, F -measure), are (0.812, 0.861, 0.832), (0.668, 0.732, 0.686), and (0.927, 0.956, 0.939), respectively.

Besides, the improvement of prediction performance obtained by transfer learning is different in classifiers. SGD classifier has the greatest improvement of prediction results for HTTPD and AXIS projects, i.e., (10.28%, 24.22%, 18.49%) and (7.97%, 19.48%, 13.81%), respectively. For the MySQL project, GBC classifier achieves the greatest improvements after using transfer learning, i.e., (16.78%, 23.03%, 18.89%).

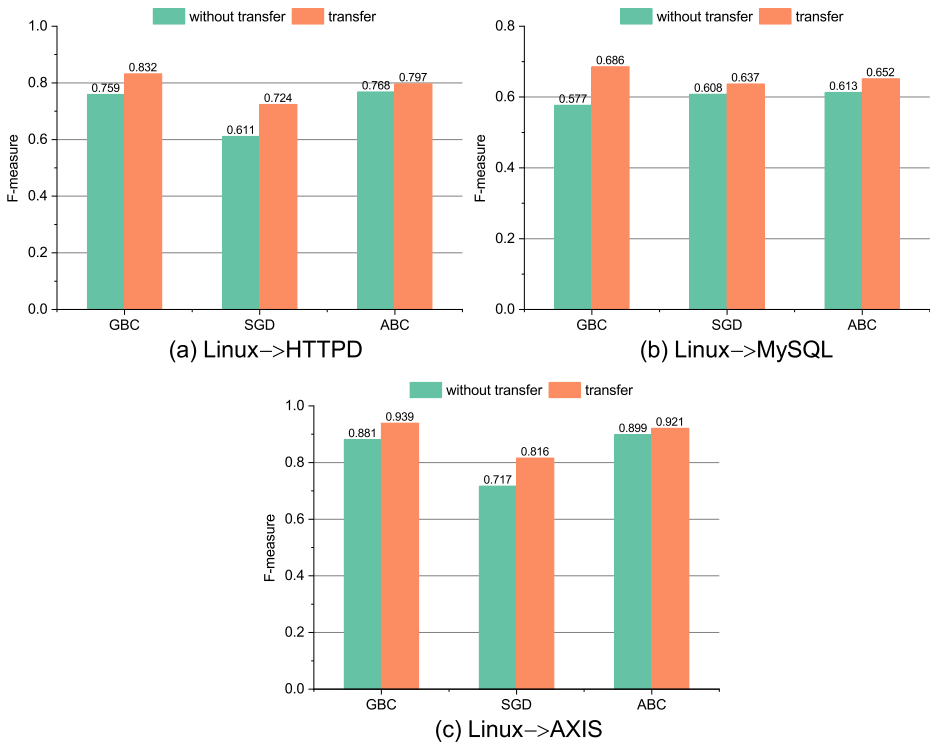


Fig. 4 Transferring Linux to predict bug types for MySQL, HTTPD and AXIS. **a** Linux→HTTPD, **b** Linux→MySQL, and **c** Linux→AXIS

Finding no. 1 *The results of cross-project bug type prediction using transfer learning are better than those without transfer learning, and the prediction results obtained by the GBC classifier are the best.*

Implication *It is practical to use the proposed framework to perform the cross-project bug type prediction. Since GBC classifier achieves the best prediction performance, it is suggested to use GBC classifier for the cross-project bug type prediction.*

4.2 The impact of different source projects

To further analyze the impact of transfer learning between different projects, we conduct experiments by exchanging the source and target projects. Table 4 shows six pairs of source and target projects. These pairs are generated from HTTPD, MySQL, and AXIS projects, e.g., MySQL→HTTPD and AXIS→HTTPD. The predicted results are shown in Tables 5, 6, and 7, respectively.

As depicted in Table 5, using MySQL or AXIS projects as the source project improve the prediction results of the HTTPD project, compared with prediction without transfer learning. The average improvements of prediction results, i.e., (precision, recall, F -measure), are (6.36%, 8.57%, 8.42%) and (10.93%, 16.87%, 15.01%), respectively. In addition, the prediction results obtained by using AXIS are better than those using MySQL. The phenomenon means that AXIS is more suitable as the source project than MySQL when

Table 3 Transferring Linux to predict bug types of HTTPD/MySQL/AXIS

	Classifier	Transfer or not	Precision	Recall	F-Measure
HTTPD	GBC	Without transfer	0.755	0.777	0.759
		Transfer	0.812	0.861	0.832
		<i>Impro.</i>	<i>7.55%</i>	<i>10.81%</i>	<i>9.62%</i>
	SGD	Without transfer	0.652	0.640	0.611
		Transfer	0.719	0.795	0.724
		<i>Impro.</i>	<i>10.28%</i>	<i>24.22%</i>	<i>18.49%</i>
	ABC	Without transfer	0.761	0.787	0.768
		Transfer	0.793	0.806	0.797
		<i>Impro.</i>	<i>4.20%</i>	<i>2.41%</i>	<i>3.78%</i>
MySQL	GBC	Without transfer	0.572	0.595	0.577
		Transfer	0.668	0.732	0.686
		<i>Impro.</i>	<i>16.78%</i>	<i>23.03%</i>	<i>18.89%</i>
	SGD	Without transfer	0.610	0.613	0.608
		Transfer	0.586	0.764	0.637
		<i>Impro.</i>	<i>-3.93%</i>	<i>24.63%</i>	<i>4.77%</i>
	ABC	Without transfer	0.603	0.642	0.613
		Transfer	0.646	0.677	0.652
		<i>Impro.</i>	<i>7.13%</i>	<i>5.45%</i>	<i>6.36%</i>
AXIS	GBC	Without transfer	0.883	0.885	0.881
		Transfer	0.927	0.956	0.939
		<i>Impro.</i>	<i>4.98%</i>	<i>8.02%</i>	<i>6.58%</i>
	SGD	Without transfer	0.765	0.729	0.717
		Transfer	0.826	0.871	0.816
		<i>Impro.</i>	<i>7.97%</i>	<i>19.48%</i>	<i>13.81%</i>
	ABC	Without transfer	0.894	0.909	0.899
		Transfer	0.917	0.929	0.921
		<i>Impro.</i>	<i>2.57%</i>	<i>2.20%</i>	<i>2.45%</i>

Italicized entries indicate that using transfer learning improves the predictions and how much is improved compared to not using transfer learning

Table 4 Pair setting

Pair no.	Source project	Target project
1	MySQL	HTTPD
2	AXIS	
3	HTTPD	MySQL
4	AXIS	
5	HTTPD	AXIS
6	MySQL	

Table 5 Transferring MySQL/AXIS to predict bug types of HTTPD

Classifier	MySQL→HTTPD				AXIS→HTTPD			
	GBC	SGD	ABC	Avg.	GBC	SGD	ABC	Avg.
Precision	0.777	0.769	0.762	0.769	0.827	0.761	0.819	0.802
<i>Impro.</i>	<i>2.91%</i>	<i>17.94%</i>	<i>0.13%</i>	<i>6.36%</i>	<i>9.54%</i>	<i>16.72%</i>	<i>7.62%</i>	<i>10.93%</i>
Recall	0.792	0.832	0.770	0.798	0.899	0.823	0.855	0.859
<i>Impro.</i>	<i>1.93%</i>	<i>30%</i>	<i>24.84%</i>	<i>8.57%</i>	<i>15.70%</i>	<i>28.59%</i>	<i>39.93%</i>	<i>16.87%</i>
<i>F</i> -Measure	0.779	0.782	0.759	0.773	0.856	0.771	0.833	0.82
<i>Impro.</i>	<i>2.64%</i>	<i>27.99%</i>	<i>-1.17%</i>	<i>8.42%</i>	<i>12.78%</i>	<i>26.19%</i>	<i>8.46%</i>	<i>15.01%</i>

Italicized entries indicate that using transfer learning improves the predictions and how much is improved compared to not using transfer learning

predicting bug types of HTTPD. Moreover, regarding the evaluation metrics, recall has the highest average improvement when using transfer learning.

Table 6 shows the bug type prediction results of using HTTPD or AXIS as the source project and MySQL as the target project. The average improvements are (2.18%, 14.59%, 6.34%) and (2.18%, 11.67%, 5.17%), respectively. The improvements of recall and *F*-measure are obvious when using transfer learning. For example, the recall and *F*-measure of MySQL prediction are improved by up to 19.83% and 12.65%, respectively.

For the target project of AXIS, the prediction results using HTTPD or MySQL are presented in Table 7. It can be observed that the improvements achieved by these two source projects are similar. The average improvements, i.e., (precision, recall, *F*-measure), is (5.55%, 7.85%, 7.57%) and (5.19%, 7.37%, 7.21%), respectively. In addition, using SGD classifier has the greatest improvement of prediction results.

Finding no. 2 For the bug type prediction of HTTPD, using AXIS as the source project is better than using MySQL. In addition, GBC obtains the best prediction results.

Finding no. 3 For the bug type prediction of MySQL, the results obtained by using HTTPD and/or as the source project are similar. Besides, GBC classifier performs better than SGD and ABC classifiers.

Table 6 Transferring HTTPD/AXIS to predict the bug type of MySQL

Classifier	HTTPD→MySQL				AXIS→MySQL			
	GBC	SGD	ABC	Avg.	GBC	SGD	ABC	Avg.
Precision	0.621	0.577	0.625	0.608	0.626	0.576	0.623	0.608
<i>Impro.</i>	<i>8.57%</i>	<i>-5.41%</i>	<i>3.65%</i>	<i>2.18%</i>	<i>9.44%</i>	<i>-5.57%</i>	<i>3.32%</i>	<i>2.18%</i>
Recall	0.713	0.706	0.703	0.707	0.708	0.712	0.648	0.689
<i>Impro.</i>	<i>19.83%</i>	<i>15.17%</i>	<i>9.50%</i>	<i>14.59%</i>	<i>18.99%</i>	<i>16.15%</i>	<i>0.93%</i>	<i>11.67%</i>
<i>F</i> -Measure	0.649	0.612	0.649	0.637	0.650	0.612	0.628	0.630
<i>Impro.</i>	<i>12.48%</i>	<i>0.66%</i>	<i>5.87%</i>	<i>6.34%</i>	<i>12.65%</i>	<i>0.66%</i>	<i>2.45%</i>	<i>5.17%</i>

Italicized entries indicate that using transfer learning improves the predictions and how much is improved compared to not using transfer learning

Table 7 Transferring HTTPD/MySQL to predict the bug type of AXIS

Classifier	HTTPD→AXIS				MySQL→AXIS			
	GBC	SGD	ABC	Avg.	GBC	SGD	ABC	Avg.
Precision	0.884	0.876	0.921	0.894	0.897	0.880	0.895	0.891
<i>Impro.</i>	<i>0.11%</i>	<i>14.51%</i>	<i>3.14%</i>	<i>5.55%</i>	<i>1.59%</i>	<i>15.03%</i>	<i>0.22%</i>	<i>5.19%</i>
Recall	0.884	0.896	0.942	0.907	0.903	0.906	0.900	0.903
<i>Impro.</i>	<i>-0.11%</i>	<i>23.93%</i>	<i>3.63%</i>	<i>7.85%</i>	<i>2.03%</i>	<i>25.31%</i>	<i>-0.99%</i>	<i>7.37%</i>
<i>F-Measure</i>	0.882	0.874	0.929	0.895	0.899	0.882	0.895	0.892
<i>Impro.</i>	<i>0.11%</i>	<i>21.90%</i>	<i>3.45%</i>	<i>7.57%</i>	<i>2.04%</i>	<i>23.01%</i>	<i>-0.33%</i>	<i>7.21%</i>

Italicized entries indicate that using transfer learning improves the predictions and how much is improved compared to not using transfer learning

Finding no. 4 For predicting bug types of AXIS, using HTTPD as the source project with ABC classifier obtains the best prediction results.

Implications The source projects, as well as the classifiers, have an impact on the prediction results when conducting transfer learning. For HTTPD, it is suggested to use AXIS as the source project and to use GBC as the classifier. For MySQL, both AXIS and HTTPD are suitable as the source project, while the classifier is recommended to use GBC. In prediction bug types of AXIS, it is suggested to use the ABC classifier and the source project HTTPD.

4.3 The impact of data size of the source project

To answer RQ3, the impact of the number of bug reports from the source project is analyzed in this part. We use Linux as the source project and randomly select a portion sample (i.e., 10%, 20%, ... , 100%) of the total number of bug reports from Linux to perform the experiments. For each experiment, MySQL, HTTPD, and AXIS are taken as the target project. Figure 5 shows the prediction results. For HTTPD and MySQL, the prediction results (i.e., precision, recall, and *F*-measure) grow with an increasing portion of data from the source project. For the AXIS project, the recall metric increases, while the precision and *F*-measure fluctuate.

Finding no. 5 Increasing the amount of data from the source project is capable of improving the prediction results of the target project.

Implication When performing cross-project bug type prediction based on transfer learning, to improve the prediction performance of the target project, it is suggested to increase the data size of the source project.

5 Threats to validity

5.1 Internal threats

Threats to internal validity come from experiments, i.e., the data sampling of training and test data. To reduce the randomness of the experiments, the final results in this paper are

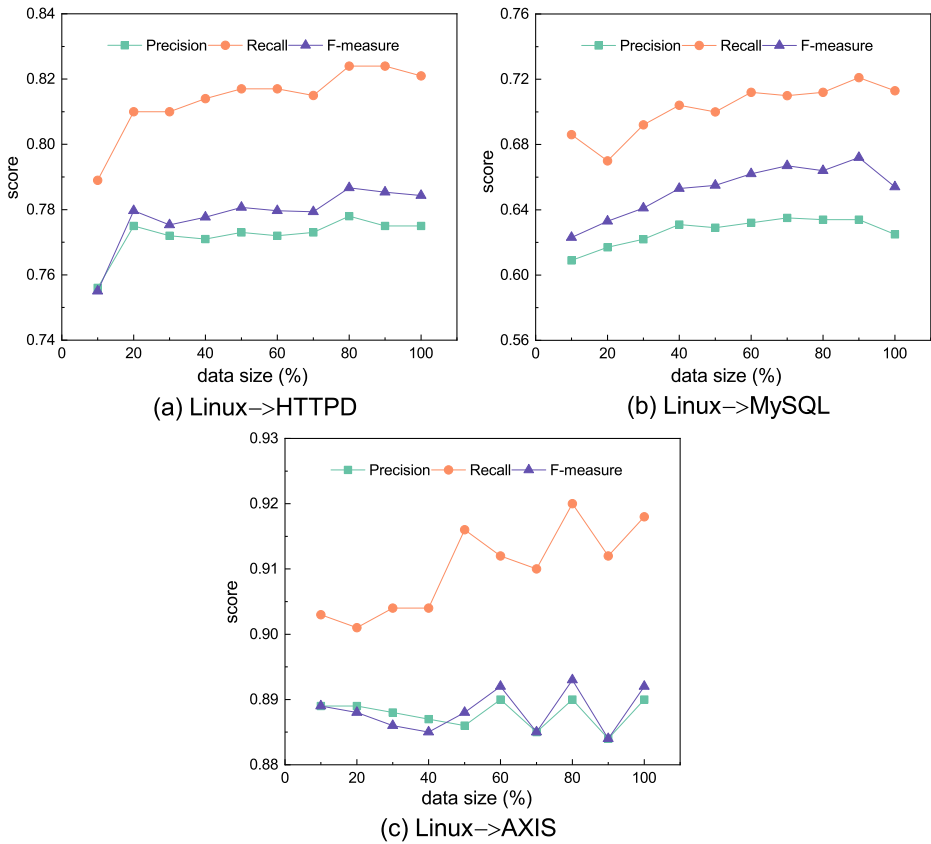


Fig. 5 The impact of the data size of the source project. **a** Linux→HTTPD, **b** Linux→MySQL, and **c** Linux→AXIS

obtained by averaging results from a total of 1000 times of experiments. A second threat is about the classifiers used for prediction. Although three different classifiers are used in the experiments, the prediction results may still be different when using other classifiers. The last threat comes from the mislabeling of bug reports. Although the classification results have been carefully checked, errors may still exist in the labeled data.

5.2 External threats

Threats to external validity come from the generalization of our results. Although we have conducted experiments on four popular open-source projects, i.e., Linux, MySQL, AXIS, and HTTPD, we do not try to claim our findings or conclusions reflect all software. The proposed framework is applicable for the prediction of other types of bugs, such as environmental-dependent bugs (Cavezza et al. 2014) and concurrency bugs (Asadollah et al. 2017).

6 Related work

Bugs are a major factor threatening the reliability of software systems (Qin et al. 2017; Qiao et al. 2018; Xiao et al. 2019). Over the past 10 years, researchers have conducted extensive studies on the prediction of software bugs from several aspects, such as predicting bug-fixing time (Bhattacharya and Neamtiu 2011), predicting bug locations (Zhang et al. 2018; Xu et al. 2019), and predicting bug severities (Kim et al. 2007). To effectively identify bugs and reduce maintenance costs of software systems, several statistical models have been developed (D'Ambros et al. 2010). Typically, these models are constructed based on historical information and they require a large amount of labeled data.

In 2008, Antoniol et al. (2008) proposed the problem of bug type classification, separating the real defects from the non-real defects. In this paper, they constructed three classifiers, including decision trees, naive Bayes, and logistic regression, to classify real defects from other issues (such as enhancement and reconstruction problems). Pingclasai et al. (2013) proposed a topic modeling method using three classifiers, i.e., decision tree, naive Bayes, and logistic regression, to classify bugs. It is found that naive Bayes is the most effective classifier. In addition, statistical and machine learning techniques were used to classify concurrency bugs (Padberg et al. 2013). Moreover, several studies conducted classification based on the triggering conditions of bugs. Frattini et al. (2016) classified bugs into environment-dependent defects and load-dependent defects based on the reproducibility characteristics of defects using naïve Bayes and Bayesian network. Xia et al. (2014) used Bayes multinomial as the classifier for the type prediction of Bohrbugs and Mandelbugs.

Although machine learning methodologies have achieved significant success in knowledge engineering work (Wu et al. 2008; Yang and Wu 2006), many machine learning methods work well only under a common assumption, i.e., the training and test data are drawn from the same feature space and the same distribution. When the assumption does not hold, traditional classification methods may perform worse. Transfer learning techniques have been applied successfully in many real-world applications. Raina et al. (2006) and Dai et al. (2007a) proposed using transfer learning techniques to learn text data cross domains, respectively. Transfer learning method SCL was used to solve NLP problems and sentiment classification problems in the work (Blitzer et al. 2006).

In addition, transfer learning has been applied to software bug prediction. For example, Nam et al. (2013) proposed a transfer defect learning approach TCA+ by extending TCA. Later, Nam and Kim (2015) presented a heterogeneous defect prediction method to match up different metrics in different projects. A unified approach was provided for both cross-project and within-project software defect predictions (Wu et al. 2018). Recently, Qin et al. (2018) proposed TLAP approach to perform aging-related bug prediction using a cross-project model. In their work, TCA was conducted to reduce the distribution difference between the training project and the target project. However, the purpose of existing work is to predict defective modules, while our work is aiming to predict the types of bugs using bug reports.

7 Conclusion and discussion

In this paper, we proposed a framework of cross-project bug type prediction based on a transfer learning method, TrAdaBoost. In the framework, labeled bug reports of a project were used to predict bug types of another project, which has insufficient labeled data to train the classification model. Four projects, i.e., Linux, MySQL, HTTPD, and AXIS, were used

to test the framework. The experimental results showed that the framework can improve the results of cross-project bug type prediction. Moreover, the impact factors of the prediction results were investigated, including the pair of source and target projects, and the data size of the source project.

In our future work, we plan to combine source code metrics (e.g., program size metrics, McCabe complexity metrics, and Halstead metrics) with bug report features to improve the prediction results. To extract source code metrics, several program analysis tools can be used. For example, SVF was developed to do scalable and precise intraprocedural static value-flow analysis for C programs (Sui and Xue 2016). SUPA, proposed by Sui and Xue (2018), focuses on performing strong updates on-demand flow and context-sensitively for analyzing C and C++ programs. There are also tools for analyzing Java and python programs (Feng et al. 2018; Gharibi et al. 2018).

Funding information This work was supported in part by the National Natural Science Foundation of China under Grant 61772055 and Grant 61872169, in part by the Technical Foundation Project of Ministry of Industry and Information Technology of China under Grant JSZL2016601B003, and in part by the State Key Laboratory of Software Development Environment under Grant SKLSDE-2018ZX-09.

References

- Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y.G. (2008). Is it a bug or an enhancement?: a text-based approach to classify change requests. In *CASCON* (vol. 8, pp. 304–318).
- Asadollah, S.A., Sundmark, D., Eldh, S., Hansson, H. (2017). Concurrency bugs in open source software: a case study. *Journal of Internet Services and Applications*, 8(1), 4.
- Bhattacharya, P., & Neamtiu, I. (2011). Bug-fix time prediction models: can we do better? In *Proceedings of the 8th Working Conference on Mining Software Repositories* (pp. 207–210): ACM.
- Blitzer, J., McDonald, R., Pereira, F. (2006). Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing, Association for Computational Linguistics* (pp. 120–128).
- Cavezza, D.G., Pietrantuono, R., Alonso, J., Russo, S., Trivedi, K.S. (2014). Reproducibility of environment-dependent software failures: an experience report. In *2014 IEEE 25th International Symposium on Software Reliability Engineering* (pp. 267–276): IEEE.
- Cotroneo, D., Grottko, M., Natella, R., Pietrantuono, R., Trivedi, K.S. (2013). Fault triggers in open-source software: an experience report. In *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)* (pp. 178–187): IEEE.
- Dai, W., Xue, G.R., Yang, Q., Yu, Y. (2007a). Co-clustering based classification for out-of-domain documents. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp 210–219): ACM.
- Dai Wenyuan, Y.Q., Guirong, X., et al. (2007b). Boosting for transfer learning. In *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, USA (pp. 193–200).
- D'Ambros, M., Lanza, M., Robbes, R. (2010). An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)* (pp 31–41): IEEE.
- Du, X., Zheng, Z., Xiao, G., Yin, B. (2017). The automatic classification of fault trigger based bug report. In *2017 IEEE International symposium on software reliability engineering workshops (ISSREW)* (pp. 259–265); IEEE.
- Feng, Y., Dreef, K., Jones, J.A., van Deursen, A. (2018). Hierarchical abstraction of execution traces for program comprehension. In *Proceedings of the 26th Conference on Program Comprehension* (pp. 86–96): ACM.
- Frattini, F., Pietrantuono, R., Russo, S. (2016). Reproducibility of software bugs. In: Principles of performance and reliability modeling and evaluation (pp. 551–565): Springer.
- Freund, Y., & Schapire, R.E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1), 119–139.

- Friedman, J.H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189–1232.
- Gharibi, G., Alanazi, R., Lee, Y. (2018). Automatic hierarchical clustering of static call graphs for program comprehension. In *2018 IEEE International conference on big data (Big Data)* (pp. 4016–4025): IEEE.
- Grottke, M., & Trivedi, K.S. (2005). A classification of software faults. *Journal of Reliability Engineering Association of Japan*, 27(7), 425–438.
- He, Z., Shu, F., Yang, Y., Li, M., Wang, Q. (2012). An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering*, 19(2), 167–199.
- Javed, M.Y., Mohsin, H., et al. (2012). An automated approach for software bug classification. In *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems* (pp. 414–419): IEEE.
- Jing, X., Wu, F., Dong, X., Qi, F., Xu, B. (2015). Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (pp. 496–507): ACM.
- Kim, S., Zimmermann, T., Whitehead, J.r. E.J., Zeller, A. (2007). Predicting faults from cached history. In *IEEE Computer Society Proceedings of the 29th international conference on Software Engineering* (pp. 489–498).
- Loper, E., & Bird, S. (2002). Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*.
- Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013a). Efficient estimation of word representations in vector space. arXiv:13013781.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp 3111–3119).
- Nam, J., Pan, S.J., Kim, S. (2013). Transfer defect learning. In *2013 35th international conference on software engineering (ICSE)* (pp. 382–391): IEEE.
- Nam, J., & Kim, S. (2015). Heterogeneous defect prediction. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering* (pp 508–519): ACM.
- Padberg, F., Pfaffe, P., Blersch, M. (2013). On mining concurrency defect-related reports from bug repositories. In *International Workshop on Mining Unstructured Data* (vol 10).
- Pan, S.J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 1345–1359.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: machine learning in python. *Journal of Machine Learning Research*, 12(Oct), 2825–2830.
- Pingclasai, N., Hata, H., Ki, M. (2013). Classifying bug reports to bugs and other requests using topic modeling. In *2013 20th asia-pacific software engineering conference (APSEC)* (vol 2, pp 13–18): IEEE.
- Plisson, J., Lavrac, N., Mladenic, D., et al. (2004). A rule based approach to word lemmatization. *Proceedings of IS-2004* (pp. 83–86).
- Qiao, Y., Zheng, Z., Fang, Y., Qin, F., Trivedi, K.S., Cai, K.Y. (2018). Two-level rejuvenation for android smartphones and its optimization. *IEEE Transactions on Reliability*.
- Qin, F., Zheng, Z., Li, X., Qiao, Y., Trivedi, K.S. (2017). An empirical investigation of fault triggers in android operating system. In *2017 IEEE 22Nd pacific rim international symposium on dependable computing (PRDC)* (pp. 135–144). IEEE.
- Qin, F., Zheng, Z., Qiao, Y., Trivedi, K.S. (2018). Studying aging-related bug prediction using cross-project models. *IEEE Transactions on Reliability* (99), 1–20.
- Raina, R., Ng, A.Y., Koller, D. (2006). Constructing informative priors using transfer learning. In *Proceedings of the 23rd international conference on Machine learning* (pp. 713–720): ACM.
- Silva, C., & Ribeiro, B. (2003). The importance of stop word removal on recall values in text categorization. In *Proceedings of the International Joint Conference on Neural Networks, 2003* (vol. 3, pp. 1661–1666): IEEE.
- Sokolova, M., Japkowicz, N., Szpakowicz, S. (2006). Beyond accuracy, F-score and roc: a family of discriminant measures for performance evaluation. In *Australasian joint conference on artificial intelligence* (pp 1015–1021): Springer.
- Sui, Y., & Xue, J. (2016). Svf: interprocedural static value-flow analysis in llvm. In *Proceedings of the 25th international conference on compiler construction* (pp 265–266): ACM.
- Sui, Y., & Xue, J. (2018). Value-flow-based demand-driven pointer analysis for C and C++. *IEEE Transactions on Software Engineering*.

- Trivedi, K.S., Mansharamani, R., Kim, D.S., Grottko, M., Nambiar, M. (2011). Recovery from failures due to mandelbugs in it systems. In *2011 IEEE 17th Pacific Rim International Symposium on Dependable Computing* (pp. 224–233): IEEE.
- Wang, C., & Mahadevan, S. (2011). Heterogeneous domain adaptation using manifold alignment. In *Twenty-Second International Joint Conference on Artificial Intelligence*.
- Weiss, K., Khoshgoftaar, T.M., Wang, D. (2016). A survey of transfer learning. *Journal of Big data*, 3(1), 9.
- Wen, W., Yu, T., Hayes, J.H. (2016). Colua: automatically predicting configuration bug reports and extracting configuration options. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)* (pp. 150–161): IEEE.
- Wu, F., Jing, X.Y., Sun, Y., Sun, J., Huang, L., Cui, F., Sun, Y. (2018). Cross-project and within-project semisupervised software defect prediction: a unified approach. *IEEE Transactions on Reliability*, 67(2), 581–597.
- Wu, X., Kumar, V., Quinlan, J.R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Philip, S.Y., et al. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1), 1–37.
- Xia, X., Lo, D., Wang, X., Zhou, B. (2014). Automatic defect categorization based on fault triggering conditions. In *2014 19th International Conference on Engineering of Complex Computer Systems* (pp. 39–48): IEEE.
- Xiao, G., Zheng, Z., Yin, B., Trivedi, K.S., Du, X., Cai, K. (2017). Experience report: fault triggers in linux operating system: from evolution perspective. In *2017 IEEE 28th international symposium on software reliability engineering (ISSRE)* (pp. 101–111): IEEE.
- Xiao, G., Zheng, Z., Jiang, B., Sui, Y. (2019). An empirical study of regression bug chains in linux. *IEEE Transactions on Reliability*.
- Xu, Y., Yin, B., Zheng, Z., Zhang, X., Li, C., Yang, S. (2019). Robustness of spectrum-based fault localisation in environments with labelling perturbations. *Journal of Systems and Software*, 147, 172–214.
- Yang, Q., & Wu, X. (2006). 10 challenging problems in data mining research. *International Journal of Information Technology & Decision Making*, 5(04), 597–604.
- Yang, X., Lo, D., Xia, X., Bao, L., Sun, J. (2016). Combining word embedding with information retrieval to recommend similar bug reports. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)* (pp. 127–137): IEEE.
- Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp 694–699): ACM.
- Zhang, X.Y., Zheng, Z., Cai, K.Y. (2018). Exploring the usefulness of unlabelled test cases in software fault localization. *Journal of Systems and Software*, 136, 278–290.
- Zhou, J.T., Pan, S.J., Tsang, I.W., Yan, Y. (2014). Hybrid heterogeneous transfer learning through deep learning. In *Twenty-eighth AAAI conference on artificial intelligence*.
- Zhou, Y., Tong, Y., Gu, R., Gall, H. (2016). Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process*, 28(3), 150–176.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Xiaoting Du received her B.Sc in Automation from Yantai University and M.Sc. from Beihang University of China. She is currently a Ph.D. candidate at Beihang University. Her research interests include software reliability, and data mining.



Zenghui Zhou is currently studying for a bachelor's degree at Beihang University. His research interests include software reliability and bug type prediction.



Beibei Yin received the Ph.D. degree from Beihang University (Beijing University of Aeronautics and Astronautics), Beijing, China, in 2010. She has been a Lecturer with Beihang University since 2010. Her main research interests include software testing, software reliability, and software cybernetics.



Guanping Xiao received the B.Sc. degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2012, and the M.Sc. degree from the Civil Aviation University of China, Tianjin, China, in 2015. He is currently working toward the Ph.D. degree with Beihang University, Beijing, China. He was a Visiting Ph.D. Student with the School of Computer Science, University of Technology Sydney, Sydney, NSW, Australia, in 2018. His research interests include software reliability and empirical software engineering.