2015 IEEE International Conference on Software Quality, Reliability and Security

Cross-project Aging Related Bug Prediction

Fangyun Qin, Zheng Zheng, Chenggang Bai, Yu Qiao
School of Automation Science and Electrical Engineering
Beihang University
Beijing China
fangyunqin, zhengz@buaa.edu.cn

Zhenyu Zhang
Institute of Software Chinese Academy of Sciences
Beijing, China
zhangzy@ios.ac.cn

Cheng Chen
School of Automation Science and Electrical Engineering
Beihang University
Beijing China
clarkchenc@gmail.com

***Abstract*—In a long running system, software tends to encounter performance degradation and increasing failure rate during execution, which is called software aging. The bugs contributing to the phenomenon of software aging are defined as Aging Related Bugs (ARBs). Lots of manpower and economic costs will be saved if ARBs can be found in the testing phase. However, due to the low presence probability and reproducing difficulty of ARBs, it is usually hard to predict ARBs within a project. In this paper, we study whether and how ARBs can be located through cross-project prediction. We propose a *transfer learning based aging related bug prediction approach* (TLAP), which takes advantage of transfer learning to reduce the distribution difference between training sets and testing sets while preserving their data variance. Furthermore, in order to mitigate the severe class imbalance, class imbalance learning is conducted on the transferred latent space. Finally, we employ machine learning methods to handle the bug prediction tasks. The effectiveness of our approach is validated and evaluated by experiments on two real software systems. It indicates that after the processing of TLAP, the performance of ARB bug prediction can be dramatically improved.**

Keywords—aging related bug; cross-project; bug prediction; software aging; transfer learning

I. INTRODUCTION

Software aging, a phenomenon which behaves as increasing failure rate and progressive degradation in the long running software system, has been arising consistent attention in both academic and industrial fields since its first systematical investigation and acceptance nineteen years ago [1]. Although software aging is a progressive phenomenon, its influence couldn't be neglected. Occasional system down caused by software aging will not only cause economic loss, but also influence company credibility. Moreover, software aging can even result in the loss of human lives in the safety critical system [2].

In order to counteract the influence of software aging, a mechanism named software rejuvenation is explored. Although it has been demonstrated that rejuvenation is an effective strategy to mitigate the effects and costs of ARBs (Aging Related Bugs, the bugs contributing to the phenomenon of software aging [3]) during runtime phase [4][5][6], more effects and costs will be saved if ARBs are avoided during the testing phase in advance.

Bug prediction may help testers concentrate on the parts that are probable to be buggy. Nevertheless, it is usually difficult to predict ARBs within-project for the following reasons. (1) Within-project bug prediction may reach a good result if sufficient training data are obtained for building the prediction model. However, ARBs occupy only a small part in a project [3], and there may not have enough available data to build the prediction model. (2) The low percentage of ARBs leads to severe class imbalance problems in prediction [7]. The rare class (ARB prone class) gets less attention than non-ARB prone class, which has negative influence on the prediction. It makes the prediction of ARBs difficult.

Based on above presentations, two interesting problems arise:

(1) In order to get sufficient training data, can we use cross-project data to make ARB prediction?

(2) If yes, how to do it?

Cross-project bug prediction can help to obtain enough training data from a different project to build the prediction model. The driving research hypothesis of this work is that some software features, such as the software complexity, its size, the programming structures related to resource management, and other features, might be related to the presence of ARBs, no matter in the same project or not [10]. Even from different projects, the types of ARBs are mainly memory bloating and leaking, unreleased file-locks, unterminated threads, storage fragmentation, data corruption, accumulation of round-off errors and so on. Thus, the ARBs may have similar features, which make their cross-project prediction feasible. However, on the other hand, it's a serious challenge to get good prediction results for cross-project bug prediction [8]. The main difficulty comes from the different data distributions among the projects, so traditional machine learning methods may not work well when they are conducted directly on the training data set.

Transfer learning is a method that could reduce the distribution difference between different objects. It has the latent ability to explore the common characteristics between training and testing projects to improve the prediction performance [9]. However, different from the prediction of bugs with frequent appearance, the severe class imbalance is another challenge for the cross-project ARB prediction [7].

To solve the cross-project ARB prediction problem, a *transfer learning based aging related bug prediction approach* (TLAP) is proposed in this paper. It uses metrics proposed in [10]. Based on these metrics, a latent space is found where distance between the training sets and testing sets is minimized [11]. In the latent space, the distributions of the two sets are close, so traditional binary classification method can be used. In addition, we use class imbalance learning to reorganize the set in order to mitigate the class imbalance effects. To verify the proposed method, we conduct experiments on two real complex software systems: Linux and MySQL. We compare the prediction performance with and without transfer learning under three different machine learning methods.

The rest of the paper is organized as follows. Section II presents the background, followed by the approach in detail in Section III. In Section IV, experiments are presented and threats to validity are discussed in Section V. Section VI concludes this study and describes future work.

II. BACKGROUND

From the time when software aging was systematically investigated [1], more and more researchers have been throwing themselves into this area.

The bugs contributing to the phenomenon of software aging are defined as Aging Related Bugs (ARBs) [3][12]. Due to their influence on the real system, software rejuvenation is proposed to mitigate the influence of software aging. It tries to occasionally terminate the system, clean its internal state and restart to release system resources so that software performance can be recovered [4].

Bug prediction is a popular research area in software engineering. Researchers struggle to predict the location of bugs through establishing relationship between bug existence and software metrics. Machine learning approaches are usually used to handle the prediction.

However, there are few works studying the prediction or even mechanism of ARBs. Grottke et al. [12] discussed the physics of aging related failures and summarized basic concepts and foundation of software aging. Cotroneo et al. [13] explored the relationship between several static software metrics and software aging. Furthermore, they tried to predict ARBs using traditional machine learning methods [10] and achieved good prediction results in within-project prediction.

Different from traditional bug prediction, the low proportion of ARBs leads to the difficulty in obtaining enough training data for within-project bug prediction. Thus, Cross-project bug prediction is necessary for the prediction of ARBs.

Transfer learning is a method that is proposed to deal with the cross-project bug prediction [9]. It makes the transferred sets close to each other [9]. Then traditional machine learning methods can be back in service based on the transferred sets.

Pan et al. [11] proposed a method named Transfer Component Analysis (TCA) that transfers both the training sets and testing sets into a latent space using maximum mean discrepancy. In the latent space, difference between two sets was reduced while the data variance was preserved. In [14], Nam et al. extended TCA to TCA+ by adding normalization selection before performing TCA. Experiments showed that TCA+ significantly improved the prediction performance in cross-project bug prediction.

However, unlike bugs with frequent appearance [11], ARBs' complex activation and/or propagation conditions lead to their low percentage. Many transfer learning methods proposed for cross-project bug prediction may not work well with respect of cross-project ARBs prediction. As a result, more details should be taken into consideration for the prediction problem.

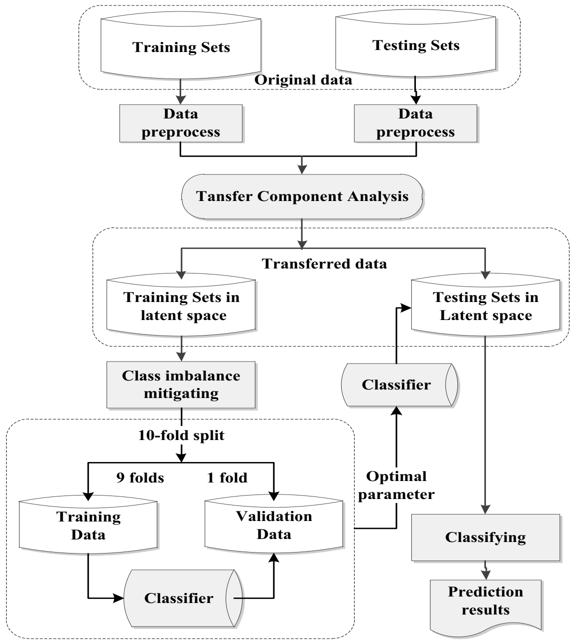

Fig. 1 Aging Related Bug prediction process

III. TRANSFER LEARNING BASED ARB PREDICTION APPROACH

In this section, we propose a transfer learning based aging related bug prediction approach (TLAP) to take the cross-project ARB prediction. TLAP requires painstaking parameter settings to build an appropriate model for the data in our case. Therefore, the training data after class imbalance mitigating procedure are randomly split into ten folds according to the bug proportion. Then nine folds (training data) are used to build the model, and one fold is used as validation data. On the same training data, our method is conducted several times with different parameters. Then optimal parameter is obtained from the prediction performance on the validation data. With the optimal parameter, the final adopted model is applied to the testing sets. In our experiment, each case is repeated for ten times in order to avoid the occasional situation. The flow chart of the approach is shown in Fig. 1. In the following, we will present each part in detail.

A. Data Preprocess

Let $X_{tr} = \{(x_{11}, \dots, x_{1p}, c_1)^T, \dots, (x_{n_1 1}, \dots, x_{n_1 p}, c_{n_1})^T\}$ and $X_{te} = \{(x_{11}, \dots, x_{1p})^T, \dots, (x_{n_2 1}, \dots, x_{n_2 p})^T\}$ be training and testing sets respectively, where x_{ij} is the jth metric value of sample x_i and n_1 and n_2 are the number of samples in each set. Meanwhile, p is the number of metrics extracted in each project, and c_i is the corresponding class of each sample.

Normalization is very useful for improving the performance of a classifier by assigning all metrics in a data set an equal weight. For each metric value x_{ij} of sample x_i, we have

$$x'_{ij} = (x_{ij} - mean(x_{.j}))/std(x_{.j}) \quad (1)$$

where x'_{ij} is the corresponding value after normalization, $mean(x_{.j})$ is the mean value of the jth metric and $std(x_{.j})$ is the corresponding standard deviation in each project. This normalization is applied to both training and testing sets.

B. Transfer Component Analysis

Transfer Component Analysis (TCA) is a transfer learning method which is able to reduce the distance between training sets and testing sets and preserve data properties simultaneously [11]. The method is graphically shown in Fig. 2 and described in detail as follows.

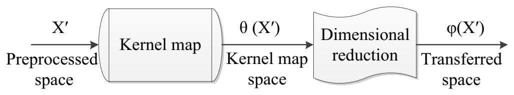

Fig. 2 The data process flow of TCA

Assume $X' = [X'_{tr}\ X'_{te}]$, and that θ is the kernel map function that transforms the sample into the kernel map space.

$$K = \begin{bmatrix} K_{tr,tr} & K_{tr,te} \\ K_{te,tr} & K_{te,te} \end{bmatrix} = (\theta(X'))^T \theta(X') = (K^{\frac{1}{2}})^T K^{\frac{1}{2}} \quad (2)$$

where K is the kernel matrix, $K_{tr,tr}$ and $K_{te,te}$ are the kernel matrices in the training and testing sets respectively. $K_{tr,te}$ and $K_{te,tr}$ show the relationship between training and testing sets. In our case, Laplacian kernel $k(x'_i, x'_j) = exp(-|x'_i - x'_j|/\tau)$ is used, where τ is a tradeoff parameter.

We use the distance between data means as the distance between training and testing sets. Thus, their distance in the kernel map space is $Dist(\theta(X'_{tr}), \theta(X'_{te})) = tr(KL)$, where $L_{ij} = 1/n_1^2$ if $x'_i, x'_j \in X'_{tr}$, $L_{ij} = 1/n_2^2$ if $x'_i, x'_j \in X'_{te}$, otherwise $L_{ij} = -1/n_1 n_2$.

In addition, a matrix P is used to shift the samples from the kernel map space to an m ($m < n_1 + n_2$) dimensional space (transferred space). Assume the comprehend function that transfers data from preprocessed space to the transferred space is φ, then we have $(\varphi(X'))^T \varphi(X') = (P\theta(X'))^T P\theta(X') = KSS^T K$, where $S = K^{-1/2} P^T$. From above equations, it is obtained that the corresponding data in the transferred space is $\varphi(X') = S^T K$. In the transferred space, the distance between two sets is $Dist(\varphi(X'_{tr}), \varphi(X'_{te})) = tr(S^T KLKS)$.

The minimum distance between above two sets is $min\, tr(S^T KLKS)$. However, minimizing the distance between objects alone couldn't reach a good prediction performance. Keeping the data variance in the transferred space is helpful to improve prediction performance. The data variance in the transferred space is $Var(\varphi(X')) = S^T KZKS/(n_1 + n_2)$, where $Z = (I_{n_1+n_2} - 1/(n_1+n_2)\,\mathbf{1}_{(n_1+n_2)\times(n_1+n_2)})$, $I_{n_1+n_2}$ is the identity matrix and $\mathbf{1}_{(n_1+n_2)\times(n_1+n_2)}$ is a matrix in which all elements equal to 1. Taking the data variance into consideration, the above kernel learning problem can be regarded as

$$\begin{gathered} min(tr(S^T KLKS) + \lambda\, tr(S^T S)) \\ s.t.\ S^T KZKS = I_m \end{gathered} \quad (3)$$

where λ is a tradeoff parameter that controls the regularization term $tr(S^T S)$. Finally, S can be solved as the m leading eigenvectors of $(KLK + \lambda I_{n_1+n_2})^{-1} KZK$.

C. Class Imbalance Mitigating Procedure

Class imbalance learning mainly concentrates on the case where some classes of data are severely under-represented when compared to other classes [15]. By convention, the under-represented class is regarded as minority class and correspondingly the class whose size is larger is called the majority class. The challenge of class imbalance learning is that the minority class can't draw same attention as majority class [16].

Assuming D is a data set with N samples. D_{min} includes examples belonging to the minority class (ARB prone class in our case), and its size is N_{min}. At the same time, D_{max} is made up of samples pertaining to the majority class with size N_{maj}. Here, we define the size ratio between the majority and minority classes as $\rho = N_{maj}/N_{min}$.

Let O_{min} and O_{maj} denote the number of two classes after class imbalance process respectively. Here, we keep the number of majority class as before, that is $O_{maj} = N_{maj}$. Meanwhile, we enlarge the minority class in the transferred space in order to let it get more attention by the classifier. Here we set $O_{min} = \lfloor 1.5 * \rho * N_{min} \rfloor$.

D. Machine Learning Methods for Classification

The target unit in our prediction problem is file. It is a proper granularity to allow researchers to find the bug location through analysis. Furthermore, we regard the problem of ARBs location as a binary classification problem. We do not care the number of ARBs in a file, but their existence. If there is at least one ARB in the file, we mark the file as ARB prone. Otherwise, it is labeled as ARB free.

Machine learning methods have been successfully used in many applications to solve classification problems. Different machine learning methods and their learned classifiers may have different assumptions about the data and reach different performance on the same data set. In this paper, we implement TLAP with three popular machine learning methods.

1) Naive Bayes

Naive Bayes (NB) is a widely used machine learning algorithm [17]. The posterior probability of the given sample is written as

$$P(C_j|M) = \frac{[\prod_{i=1}^{m} P(M_i|C_j)]P(C_j)}{P(M)} \quad (4)$$

where m is the number of features, M is the representation of all the features after class imbalance mitigating procedure and M_i presents the *i*th feature. Meanwhile, C_j presents the *j*th class. The predicted class of the sample belongs to the class which has the largest value of $P(C_j|M)$.

2) Logistic Regression

Logistic regression (LR) [17] sets the value of dependent variable between 0 and 1 through sigmoid function, which helps to avoid too big or too small data in the calculation result. The logistic regression function can be written as

$$P(M) = \frac{1}{1 + e^{-(a_0 + a_1 M_1 + \cdots + a_m M_m)}} \quad (5)$$

where m is the number of the features and M_i (*i=1, 2, ... , m*) presents the *i*th feature, a_i (*i=1, 2, ... , m*) is the coefficient of corresponding feature. In our case, if $P(M)$ is larger than 0.5, the sample is classified as ARB prone, otherwise, it is classified as ARB free.

3) K-nearest Neighbor

K-nearest neighbor (KNN) is a typical intuitive machine learning method [17]. The samples belonging to the same class must have close relationship. Given a new sample, the method looks for its K-nearest neighbors and records their classes. The class which has the largest number of samples is set to the class of the given sample. It can be presented as follows:

$$g(M) = arg \max_{j=1,\cdots,c} \pi_j(M) \quad (6)$$

where $\pi_j(M)$ is the number of samples belong to class *j*. *c* is the number of classes, here *c=2*.

E. Software Complexity Metrics

The relationship among different software components is complex. It is usually difficult and impractical to clarify all the traits in each part to get the prediction model. In our approach, static metrics are used, since they are relatively easy to get and reflect certain characteristics of a software system.

The metrics we use are program size, McCabe's cyclomatic complexity, Halstead metrics and Aging-Related Metrics [10]. The first three categories of metrics are commonly used static metrics in software reliability. Detailed information about the meaning of each metric can be found in [18]. Aging-Related Metrics are proposed by Cotroneo et al. [10], which can improve the ARB prediction performance.

IV. Experiments

In this section, we manage to verify our approach on two real complex systems. We describe the data sets, evaluation metrics and experimental results in detail. To the best of our knowledge, this is the first work to predict ARBs with transfer learning.

A. Data Sets

In order to test the performance of our model, we conduct experiments on two real software systems: Linux and MySQL. Because it is impractical to extract all the subsystems in a project, we choose four important subsystems in Linux project and three in MySQL as described in TABLE I [10].

TABLE I. ARBs Information in This Study

Project	*Subsystem*	*ARBs*	*Files*	*ARB-prone files*	*% ARB-prone files*
Linux	Network Drivers	9	3400	20	0.59%
	SCSI Drivers	4			
	EXT3 Filesystem	5			
	Networking/IPv4	2			
MySQL	InnoDB Storge Engine	6	730	41	5.6%
	Replication	5			
	Optimizer	5			

As to ARBs, we use the classification criteria in [3], in which authors classified bugs into Bohrbugs, ARBs and non-aging-related Mandelbugs. Bohrbugs refer to bugs that are easily to be isolated and their activation and error propagation lack complexity. Mandelbugs are those whose activation or error propagation is complex. The complexity can be regarded as a time lag between the fault activation and the occurrence of failure, or the bugs are influenced by indirect factors. The indirect factors could be interaction with the system environment, inputs time, sequence of operations and so on. Meanwhile, Mandelbugs can be further divided into ARBs and non-aging related bugs. In our prediction approach, 1 represents for ARB prone, and 0 represents for ARB free.

TABLE I lists the ARBs information used in this study [10]. It points out that ARB prone files make up relatively a small part among the files we analyzed, especially in Linux, only 0.59 percent.

Data are collected from the bug repositories [19][20]. Cotroneo et al. [10] set the search criteria as CODE_FIX to the resolution part and looked for corresponding subsystems in the restricted time period. For Linux project, the time span is from 2003.12.01 to 2011.5.31 in version 2.6 and MySQL from 2006.08.01 to 2011.2.28 in version 5.1. The files are labeled according to the Appendix in [10].

B. Evaluation Metrics

Since ARB prone files constitute only a small part in the project, we do not use accuracy, precision, F-measure that are commonly adopted by most papers. Because these metrics work as poor indicators in the case where target class is rare [1]. We use *PD*, *PF*, and *Bal* as evaluation metrics because they are widely used in the class imbalance case [1]. Before the calculation of each measure, a confusion matrix is given in . Based on the confusion matrix, each measure can be denoted as below.

PD: it represents the percentage that an ARB prone file is predicted as ARB prone. A high value indicates many ARB

TABLE II. Confusion Matrix

		Prediction class	
		ARB prone	***ARB free***
Actual class	ARB prone	TP	FN
	ARB free	FP	TN

TABLE III DIFFERENT METHODS WITH /WITHOUT TLAP

classifier	Linux→MySQL			MySQL→Linux		
	PD	***PF***	***Bal***	***PD***	***PF***	***Bal***
NB	0.122	0.044	0.378	0.650	0.146	0.732
LR	0	0	0.293	0	0	0.293
KNN	0	0	0.293	0.05	0.006	0.328
Avg. 1	0.041	0.015	0.321	0.233	0.051	0.451
NB+TLAP	0.678±0.038	0.215±0.016	**0.725±0.016**	0.962±0.020	0.445±0.006	0.684±0.004
LR+TLAP	0.765±0.095	0.261±0.093	**0.735±0.030**	0.941±0.076	0.439±0.099	**0.674±0.068**
KNN+TLAP	0.067±0.056	0.029±0.006	**0.340±0.039**	0.668±0.237	0.261±0.091	**0.681±0.138**
Avg. 2	0.503±0.380	0.168±0.123	**0.600±0.225**	0.857±0.165	0.382±0.104	**0.680±0.005**

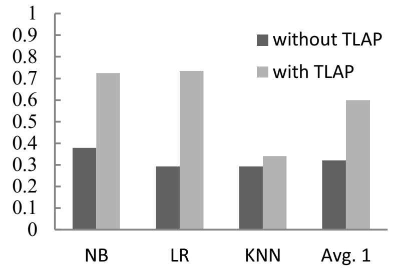

Fig. 3 The comparison of Bal when data extracted from Linux are used as training sets

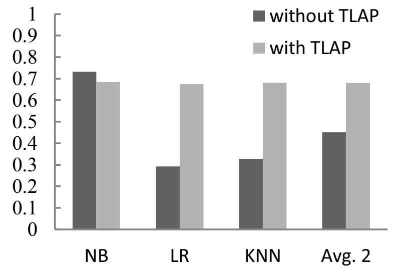

Fig. 4 The comparison of *Bal* when data extracted from MySQL are used as training sets

prone files are predicted right. It is defined as $PD = TP/(TP + FN)$.

PF: it denotes the percentage that an ARB free file is predicted as ARB prone. A low value indicates a minority of ARB free files are predicted as ARB prone. It is defined as $PF = FP/(FP + TN)$.

Bal: it works as an indicator that reflects the comprehensive behavior of the classifier [1]. *PD* and *PF* only manifest a part of the classifier performance, and are often difficult to reach desired value simultaneously. Generally, a high value of *PD* accompanies with a high value of *PF*. *Bal* trades off above two indicators and gives a sophisticated evaluation of the classifier. It is defined as $Bal = 1 - \sqrt{(0 - PF)^2 + (1 - PD)^2}/\sqrt{2}$.

C. Experimental Results

In this part, we attempt to investigate the performance of our model. The prediction performance with and without our model *transfer learning based ARB prediction approach* (TLAP) is listed, along with three different classifiers.

TABLE III, Fig. 3 and Fig. 4 show the predicted result of *PD, PF, Bal* in different ways. *PD* refers to the probability that an ARB prone file is predicted right, the higher the score, the better. *PF* represents the probability that an ARB free file is predicted as ARB prone. The lower the value of *PF* is, the better. *Bal* combines *PD* and *PF* and denotes the Euclidean distance from the ideal objective *PD*=1 and *PF*=0. It's a comprehensive evaluation metric. The higher the value is, the better the prediction performance of the model. In the table and figures, NB, LR and KNN represent the naive Bayes classifier, logistic regression, and K-nearest neighbor method respectively. Avg. 1 is the mean of results in three plain cases, and Avg. 2 is the mean after TLAP is added. From the table and figures, we can have the following three conclusions:

(1) TLAP can improve cross-project ARBs prediction performance

Bal values having an improvement when performed with TLAP are in boldface. From the table and figures, it is obvious that classifiers conducted upon TLAP work better than traditional machine learning methods performed directly on plain data set. When data extracted from Linux are used as training sets, the mean of *Bal* without TLAP is only 0.321. After TLAP is conducted, the mean of *Bal* improves to 0.600. On the other side, when data extracted from MySQL are used as training sets, the mean of *Bal* rises from 0.451 to 0.680 once TLAP is performed.

On the plain set, the distributions of Linux and MySQL are largely different. Traditional machine learning methods lose their function when they are directly used on the plain data. Moreover, the small constitution of ARB prone files in the project increases the prediction difficulty. When sets are processed with normalization, TCA, and class imbalance learning step by step, the distribution difference between the two sets is reduced, and minority class obtains more attention from the classifier. After utilizing the method of TLAP, although *PF* increases along with *PD*, its influence on *Bal* is less than the increase in *PD*.

(2) The prediction performance after TLAP is stable

As shown in TABLE III, the prediction performance is stable in each classification situation. Among the three classifiers conducted on TLAP, NB+TLAP outperforms. When data from Linux are used as training sets, the standard deviation in *PD* is 0.038, in *PF* is 0.016, and in *Bal* is 0.016. At the same time, in the opposite direction, the standard deviation in *PD* is 0.020, in *PF* is 0.006, and in *Bal* is only 0.004. These standard deviations are smaller than those in other cases, which makes NB+TLAP outstand in the aspect of stable prediction performance.

(3) The percentage of ARB prone files in a project has a big influence on prediction performance when KNN is used as classifier.

TABLE III points out that the prediction result of *Bal* is very low (only 0.340) when data from Linux are used as training sets. It is clear that KNN is very sensitive to the distance between under-predicted sample and its surrounded samples. Among the files we analyze, the ARB prone files make up 5.6% in MySQL and only 0.59% in Linux [10]. When data from Linux are used as training sets, it provides only a small percentage of ARBs information. Class imbalance mitigating procedure only duplicates the number of ARB prone files, but it couldn't contribute to more information on the ARB characteristic. Only a project with higher percentage of ARB prone files could give more information of the ARB characteristic, and get more attention in the KNN.

From the experiments, we can answer the two research questions proposed in Section I as follows:

A1: In order to get enough training data, cross-project data can be used to make ARB prediction.

A2: By reducing the distribution difference among the data sets extracted from different software systems, transfer learning based ARB prediction approach can really improve the prediction performance.

V. THREATS TO VALIDITY

As other empirical experiments, there are four main threats to the validity of our study

(1) ARB mark precision

The bug information is collected on the bug reports, which are descriptions written by users. So the accuracy and completeness of the description influence the judge of bug.

(2) Open source validity

The projects we analyze are all open source projects, which show different characteristic from closed source projects.

(3) More cases need to be studied

In the work, only two real cases are studied here. It is necessary to verify the approach with more cases implemented in different programming languages.

VI. CONCLUSION AND FUTURE WORK

In this paper, we apply a new method named TLAP in the cross-project prediction of ARBs. Transfer learning can extract some common information among two different distributed objects and make their distribution close to each other. Here we take advantage of TCA proposed by Pan et al. [11] to transfer data to a latent space. By duplicating the minority class, the approach deals with the class imbalance problem in the training data, making minority class draw more attention from the machine learning methods. The experiments in Linux and MySQL show that the approach is effective to make cross-project ARB prediction.

In the future, we plan to improve our work in several parts. First, we only use two projects when validating the efficiency of the method, more projects programed in different language are suggested. Second, some aging related metrics will be explored in order to improve the precision of the prediction. Finally, more methods on class imbalance learning will be explored to improve the prediction precision on ARBs.

REFERENCES

[1] Y. Huang, C. Kintala, N. Kolettis, and N.D. Fulton, "Software rejuvenation: Analysis, models, and applications", In Proc. of 25th Int. Symposium on Fault-tolerance Computing, pp. 381-390, June 1995.

[2] E. Marshall, "Fatal error: how patriot overlooked a scud," Science, pp. 1347-1347, 1992.

[3] M. Grottke, A.P. Nikora, and K.S. Trivedi, "An empirical investigation of fault types in space mission system Software," Proc. IEEE/IFIP Int'l Conf. Dependable Systems and Networks, pp. 447-456, 2010.

[4] K. Vaidyanathan and K.S. Trivedi, "A comprehensive model for software rejuvenation," IEEE Trans. Dependable and Secure Computing, vol. 2, no. 2, pp. 124-137, 2005.

[5] L. Zhao, Q. Song, and L. Zhu, "Common software-aging-related faults in fault-tolerant systems," CIMCA'2008, pp. 327-331, Dec. 2008.

[6] R. Matias and P. J. F. Filho, "An experimental study on software aging and rejuvenation in web servers," 30th IEEE COMPSAC Washington, DC, USA, pp. 189-196, Sept. 2006.

[7] S. Wang, and X. Yao, "Using class imbalance learning for software defect prediction," IEEE Trans. Rel., vol. 62, pp. 434-443, June 2013.

[8] T. Zimmermann, N. Nagappan, H. Gall, E. Giger and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp. 91-100, 2009.

[9] S. J. Pan and Q. Yang, "A survey on transfer learning," IEEE Trans. Knowledge and Data Eng., vol. 22, pp. 1345–1359, Oct. 2010.

[10] D. Cotroneo, R. Natella, and R. Pietrantuono, "Predicting aging-related bugs using software complexity metrics," Performance Evaluation, vol. 70, pp. 163-178, March 2013.

[11] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," IEEE Trans. Neural Netw., vol. 22, pp. 199-210, Feb. 2011.

[12] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," Proc. First Int'l Workshop Software Aging and Rejuvenation, pp. 1-6, Nov. 2008.

[13] D. Cotroneo, R. Natella, and R. Pietrantuono, "Is software aging related to software metrics?," Proc. Second Int'l Workshop Software Aging and Rejuvenation, pp. 1-6, Nov. 2010.

[14] J. Nam, S.J. Pan, and S. Kim, "Transfer defect learning," Proc. 2013 International Conference on Software Engineering, pp. 382-391, May 2013.

[15] H. He and E.A. Garcia, "Learning from imbalanced data," IEEE Trans. Knowledge and Data Eng., vol. 21, pp. 1263-1284, Sept. 2009.

[16] G. M. Weiss, "Mining with rarity: a unifying framework," ACM SIGKDD Explorations Newsletter, vol. 6, no. 1, pp. 7-19, June 2004.

[17] S. Theodoridis and K. Koutroumbas. Pattern Recognition, 4th ed., Elsevier Inc., 2009.

[18] https://scitools.com/

[19] Linux kernel: http://bugzilla.kernel.org

[20] MySQL DBMS: http://bugs.mysql.com.

[21] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 2–13, Jan. 2007.