MACHINE LEARNING ASSISTED SOFTWARE DEFECT PREDICTION

ZHOU XU

PhD

The Hong Kong Polytechnic University

This programme is jointly offered by The Hong Kong Polytechnic University and Wuhan University

2021

The Hong Kong Polytechnic University
Department of Computing
Wuhan University
School of Computer Science

Machine Learning Assisted Software Defect Prediction

Zhou Xu

A thesis submitted in partial fulfilment of the requirements

for the degree of Doctor of Philosophy

December 2020

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Zhou Xu (Name of student)

Abstract

Software products have been integrated into every aspects of our daily life. However, due to various factors in the process of software design, development and configuration, the defects are inevitable in the software. The defect hidden in the software module (a code snippet) threatens the security and decrease the reliability of the software products. Therefore, it is essential to detect and fix the defective modules before delivering the products. However, due to the continuous growth of the software scale and complexity, it is an increasingly challenging task for software developers and testers to improve the software quality.

As the limited testing resources are usually unaffordable for supporting thorough code reviews, this requests a prioritization to better analyze the software product. In other words, developers and testers should reasonably allocate the limited resources to test the modules that have a high probability to contain defects. To seek for such prioritization, researchers propose software defect prediction to identify such high-risk modules for priority inspection. The most widely studied defect prediction methods are supervised models which first train a classification model on labeled software modules and then use it to determine whether or not the unlabeled modules contain defects. The supervised models need the labeled modules of historical data of the current project or external projects as the training set. According to the different sources of the training set, supervised defect prediction can be divided into the inner version defect prediction scenario, cross version

defect prediction scenario, and cross project defect prediction scenario. In the three kinds of scenarios, the training set comes from the same version of a project, the previous version of a project, and other external projects, respectively. This thesis mainly studies new machine learning technologies to solve the different difficulties faced in the three kinds of defect prediction scenarios, aiming to further improve the performance of defect prediction. The research contents are described as follows:

(1) In order to learn more discriminative feature representation and solve the inherent class imbalance problem of defect data, this thesis proposes an inner version defect prediction framework which combines a kernel principal component analysis method and a weighted extreme learning machine. The framework firstly maps the training set and test set into a high-dimensional feature space separately using the kernel principal component analysis method. The feature mapping makes it easy to distinguish the modules which are linearly inseparable in the original feature space. Then the framework uses the mapped training set to construct a classification model based on a weighted extreme learning machine to predict the labels of the mapped test set. This classification model solves the class imbalance problem by assigning different weights to the defective and non-defective software modules. We conduct experiments on ten projects in the NASA dataset and five projects in the AEEEM dataset, and use six indicators to evaluate the performance of the proposed framework. The results show that the performance of our proposed inner version defect prediction framework is gererally better than its variant methods, some feature selection methods, and class imbalanced learning methods.

(2) In order to select a subset of software modules from the previous version as the training set which is optimal for the data of the current version, this thesis proposes a two-stage training subset selection framework for cross version defect prediction. This framework first uses the sparse modeling representation selection method to filter out some useless software modules and keeps the software modules that can minimize the

error of reconstructing original data. Since this process does not rely on the assistance of the software modules from the current version, it is a self-simplification stage. Then, with the participation of the data from the current version, the framework uses the dissimilarity-based sparse subset selection method to further select a subset from the selected modules in the previous stage to effectively represent the data of the current version. The model constructed with the final selected module subset is more targeted to the data of the current version. Since this process requires the assistance of the software modules from the current version, it is an auxiliary refining stage. We conduct experiments on 67 versions from 17 projects in the PROMISE dataset and also use six indicators to evaluate the performance of the proposed framework. The results show that, across a total of 50 cross-version pairs, the overall performance of our proposed cross version defect prediction framework is superior to other training subset selection methods and the variant methods based on one-stage training subset selection.

(3) In order to further narrow the distribution difference between the two cross-project data, this thesis proposes a new transfer learning based cross project defect prediction framework by introducing a state-of-the-art balanced distribution adaptive model. Unlike the previous transfer cross project defect prediction models which only considered the marginal distribution differences across data, this model comprehensively considers the marginal and conditional distribution differences across data. In addition, considering the impacts of the similarity between cross project data on the relative importance degrees of the two distribution differences, the model also assigns the weights to the two differences for adapting different cross-project pairs. We conduct experiments on five projects in the NASA dataset and five projects in the AEEEM dataset, and also use six indicators to evaluate the performance of the proposed framework. The results show that, across a total of 40 cross-project pairs, the overall performance of our proposed cross project defct prediction framework performs better than other transfer learning based and training data

filter based cross project methods.

In conclusion, this paper aims at solving difficult problems in different software defect prediction scenarios and proposing new framework models to improve the performance of defect prediction by combining new machine learning technologies. This paper expands the application of machine learning technologies in the field of software engineering and provides new solutions to the software defect prediction task, which is of great significances for software quality assurance activities.

Keywords: Software Defect Prediction; Feature Learning; Class Imbalanced Learning; Training Subset Selection; Transfer Learning

Publications

1. **Zhou Xu**, Li Li, Meng Yan, Jin Liu, Xiapu Luo, John Grundy, Yifeng Zhang, and Xiaohong Zhang, "A Comprehensive Comparative Study of Clustering-based Unsupervised Defect Prediction Models", Journal of Systems and Software (JSS), 2020: 110862.

2. **Zhou Xu**, Tao Zhang, Jacky Keung, Meng Yan, Xiapu Luo, Xiaohong Zhang, Ling Xu, and Yutian Tang, "Feature Selection and Embedding Based Cross Project Framework for Identifying Crashing Fault Residence", Information and Software Technology (IST), 2020: 106452.

3. **Zhou Xu**, Kunsong Zhao, Meng Yan, Peipei Yuan, Ling Xu, Yan Lei, and Xiaohong Zhang, "Imbalanced Metric Learning for Crashing Fault Residence Prediction", Journal of Systems and Software (JSS), 2020, 170: 110763.

4. **Zhou Xu**, Jin Liu, Xiapu Luo, Zijiang Yang, Yifeng Zhang, Peipei Yuan, Yutian Tang, and Tao Zhang, "Software Defect Prediction Based on Kernel PCA and Weighted Extreme Learning Machine", Information and Software Technology (IST), 2019, 106: 182-200.

5. **Zhou Xu**, Shuai Li, Xiapu Luo, Jin Liu, Tao Zhang, Yutian Tang, Jun Xu, Peipei Yuan, and Jacky Keung, "TSTSS: A Two-Stage Training Subset Selection

Framework for Cross Version Defect Prediction", Journal of Systems and Software (JSS), 2019, 154: 59-78.

6. **Zhou Xu**, Shuai Li, Jun Xu, Jin Liu, Xiapu Luo, Yifeng Zhang, Tao Zhang, Jacky Keung, and Yutian Tang, "LDFR: Learning Deep Feature Representation for Software Defect Prediction", Journal of Systems and Software (JSS), 2019: 110402.

7. **Zhou Xu**, Shuai Pang, Tao Zhang, Xiapu Luo, Jin Liu, Yutian Tang, Xiao Yu, and Lei Xue, "Cross Project Defect Prediction Via Balanced Distribution Adaptation Based Transfer Learning", Journal of Computer Science and Technology (JCST), 2019, 34(5): 1039-1062.

8. **Zhou Xu**, Peipei Yuan, Tao Zhang, Yutian Tang, Shuai Li, and Zhen Xia, "HDA: Cross-Project Defect Prediction via Heterogeneous Domain Adaptation with Dictionary Learning", IEEE Access, 2018, 6: 57597-57613.

9. Kunsong Zhao, **Zhou Xu***, Meng Yan, Yutian Tang, Ming Fan, and Gemma Catolino, "Just-in-Time Defect Prediction for Android Apps via Imbalanced Deep Learning Model", in *Proceedings of the 36th ACM/SIGAPP Symposium on Applied Computing (SAC)*, 2021, accepted.

10. **Zhou Xu**, Tao Zhang, Yifeng Zhang, Yutian Tang, Jin Liu, Xiapu Luo, Jacky Keung, and Xiaohui Cui, "Identifying Crashing Fault Residence Based on Cross Project Model", in *Proceedings of the 30th International Symposium on Software Reliability Engineering (ISSRE)*, 2019: 183-194.

11. **Zhou Xu**, Sizhe Ye, Tao Zhang, Zhen Xia, Shuai Pang, Yong Wang, and Yutian Tang, "MVSE: Effort- Aware Heterogeneous Defect Prediction via Multiple-View Spectral Embedding", in *Proceedings of the 19th International Conference on Software Quality, Reliability, and Security (QRS)*, 2019:10-17.

12. **Zhou Xu**, Shuai Li, Yutian Tang, Xiapu Luo, Tao Zhang, Jin Liu, and Jun Xu, "Cross Version Defect Prediction with Representative Data via Sparse Subset Selection", in *Proceedings of the 26th International Conference on Program Comprehension (ICPC)*, 2018: 132-143.

13. **Zhou Xu**, Jin Liu, Xiapu Luo, and Tao Zhang, "Cross-Version Defect Prediction via Hybrid Active Learning with Kernel Principal Component Analysis", in *Proceedings of the 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018: 209-220.

14. **Zhou Xu**, Jin Liu, Zhen Xia, and Peipei Yuan, "An Empirical Study on the Equivalence and Stability of Feature Selection for Noisy Software Defect Data", in *Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2017: 191-196

15. **Zhou Xu**, Jin Liu, Zijiang Yang, Gege An, and Xiangyang Jia, "The Impact of Feature Selection on Defect Prediction Performance: An Empirical Comparison", in *Proceedings of the 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016: 309-320.

16. **Zhou Xu**, Jifeng Xuan, Jin Liu, and Xiaohui Cui, "MICHAC: Defect Prediction via Feature Selection Based on Maximal Information Coefficient with Hierarchical Agglomerative Clustering", in *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, IEEE, 2016, 1: 370-381.

17. Yutian Tang, Yulei Sui, Haoyu Wang, Xiapu Luo*, Hao Zhou, and **Zhou Xu***, "All Your App Links are Belong to Us: Understanding the Threats of Instant Apps based Attacks", in *Proceedings of the 28th ACM Joint European Software Engineering*

Conference and Symposium on the Foundations of Software Engineering (FSE), Accepted to appear, 2020. (co-corresponding author)

18. Xiaoyun Cheng, Naming Liu, Lin Guo, **Zhou Xu***, and Tao Zhang*, "Blocking Bug Prediction Based on XGBoost with Enhanced Features", in *Proceedings of the 44th IEEE Annual Computer Software and Applications Conference (COMPSAC)*, 2020: 902-911. (co-corresponding author)

19. Yiheng Jian, Xiao Yu*, **Zhou Xu***, Ziyi Ma, "A Hybrid Feature Selection Method for Software Fault Prediction", IEICE Transactions on Information and Systems, 2019, 102(10): 1966-1975. (co-corresponding author)

Acknowledgements

Throughout the writing of this thesis, I have received a great deal of support and assistance from the nice people I have met. Here, I would like to express my heartfelt thanks to them.

I would first like to thank my supervisor, Associate Prof. Xiapu Luo, whose expertise, valuable suggestion, in-depth discussion, and insightful feedback brought my work to a higher level and support me to complete this thesis successfully.

I would also like to acknowledge Prof. Jin Liu, my supervisor in Wuhan University. Thank him for recommending and supporting me to apply for joint PhD training program with the Hong Kong Polytechnic University. This research experience in Hong Kong has broadened my horizon and makes it possible for me to work with many domain experts, which lays a solid foundation for my future research. I will never forget this precious experience for the rest of my life.

In addition, I would like to thank many other teachers and classmates that helped me a lot during my study period. The teachers, including Jifeng Xuan, Tao Zhang, Jacky Keung, Meng Yan and Jun Xu, gave me a lot of constructive advice, helped me modify the grammatical errors and improper expressions in papers to improve their quality, and recommended the journals and conferences that are suitable for the topics of papers. The final acceptance of my papers cannot be separated from their efforts. Beside, I would also like to thank the group members in Hong Kong and mainland, including Lei Xue,

Le Yu, Yutian Tang, Xiaoqi Li, Xian Zhan, Muhui Jiang, Ningning Hou, Shuai Li, Fang Wang, Juan Li, Haoyu Luo, Pingyi Zhou, Jie Liu, Juncai Guo, Xiao Yu, Gege An, Yifeng Zhang, Zhen Xia, Shuai Pang, and Kunsong Zhao. They have created a friendly working environment for my research and often given me constant encouragement and help when I got into difficulties in my research.

Last but not least, I would like to express my special thanks to my parents, wife and daughter. My parents always offer me the best they can in my education and daily life. The company of my wife, Lixian Li, and daughter, Shulin Xu always fills me with full of passion and energy. Thanks to their love.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Background

1.1.1 The Existence and Harm of Software Defects

In recent years, with the rapid development of the information technology, the number of software products has increased sharply. Software has gradually integrated into every aspect of people's daily life and is widely applied to various fields of social, economic and military, such as mobile games, shopping software, and air control systems. Nowadays, we can say that we live in an age where software defines everything. In addition, as the breakthrough of various technologies and the increasing demand of people, many software products with complex systems and powerful functions are emerging. However, due to some uncontrolled factors, such as unclear definitions during the requirement analysis stage, imperfect schemes during the system design stage, nonstandard implementations during the coding stage and incorrect judgment during the testing stage, software systems inevitably exist defects [69]. Mohan et al. [2] have shown that the defects introduced in the software analysis and design stages account for 64% of the total defects, and the defects introduced in other development stages account for 36% of the total defects. The defects

can affect the user's experience and cause economic losses. For example, the defects in mobile games may cause the game inexplicably interrupted, and the defects in shopping software may cause the consumers being deceived. Furthermore, the defects may bring serious casualties. For example, the defects of the sequence of input data in the Therac 25 radioactive therapeutic apparatus led to the deaths of more than 10 people between 1985 and 1987.

1.1.2 Basic Concepts of Software Testing

As software can affect all aspects of human society, economy and military life, people pay more and more attention to the quality and reliability of software products. The phenomenon of low quality is common in software products, especially in large and complex software systems. How to guarantee the high quality and high reliability of software products has become a hot issue that needs to be solved urgently. This is also the goal pursued by software developers and testers. For this purpose, the software testing technology emerges. Software testing conducts operations on the program in order to find defects. Software testing generally includes four types: unit testing, integration testing, system testing, and acceptance testing [29]. More specifically, the object of unit testing is the smallest unit component of software, and the purpose is to check whether the basic unit of software is correct. The object of integration testing is the interface between software unit components, and the purpose is to check whether the interface and the function of the integrated components are correct. The object of system testing is the whole system, and the purpose is to check whether the function, performance, software and hardware environments of the system are correct. Acceptance testing is the last test before a software is delivered, and the purpose is to check whether the software system meets the original requirements. According to whether the source code is reviewed or not, software testing can be divided into white-box testing and black-box testing. The black-box testing regards

the software system as a black box and only focuses on the input and output data of the software system instead of the structure inside the box. The white-box testing involves examining and analyzing the constructs inside the box, i.e., the source code information. According to whether the program is executed or not, software testing can be divided into static testing and dynamic testing. Static testing does not need to run the code of the software system under test. It determines whether the software contains defects by checking the source code, measuring the static complexity of the program, and analyzing the flow chart of the program. Dynamic testing requires to run the code of the software system under test, and judges whether the program contains defects by comparing the output results with the predicted ones.

1.1.3 Software Defect Prediction

As the software development life cycle progresses, the cost of detecting and fixing hidden defects in the software system becomes higher. Therefore, software developers and testers want to analyze the software with the help of some testing techniques of software quality assurance, hoping to detect as many defective software modules as possible before the software is released. According to different granularities, a software module can be a file, a class or a function. In addition, considering that the software delivery date is approaching and inspecting software modules is labor-intensive and time-consuming, it is impractical to inspect all software modules with the limited and precious testing resources. In this case, developers and testers should be able to identify the software modules that are most likely to contain defects, and then allocate them test resources for checking to improve test efficiency. To achieve this goal, researchers propose **S**oftware **D**efect **P**rediction (**SDP**) techniques to accomplish this task.

In recent years, SDP has been one of the most active branches of the software

engineering research [42]. The general process of SDP is first to collect historical data (such as the source code and bug reports) of software development from version control systems (such as Subversion and Github) and bug tracking systems (such as Jira and Bugzilla). The collected data with defect labels can be used to form an initial software defect dataset. Then SDP applies different machine learning techniques to identify the defect labels of new developed software modules [156]. The software modules that are identified to contain defects with a high probability can be recommended for priority review by software developers and testers. Thus, SDP is a software quality assurance activity that can optimize test resource allocation and improve software testing efficiency [57]. In addition, from the classification perspective of software testing, SDP belongs to the static testing because it does not require to run the source code of software project.

SDP can be divided into static and dynamic SDP according to whether the time of defect occurrence is considered or not. Dynamic SDP requires to analyze the distribution of defects over time in the whole software life cycle. Static SDP constructs models on the features (such as code complexity features of software modules and change features) that measures characteristics of software defects to analyze whether other software modules contain defects.

Since static SDP is a hot topic in the cross domains between software engineering and artificial intelligence, a large number of related articles have emerged in recent years. According to the defect content differences of software modules, these articles can be divided into software defect number prediction, software defect density prediction, and software defect tendency prediction. More specifically, software defect number prediction mainly uses models to fit regression functions to calculate the number of defects contained in new modules. There are relatively few researches in this category [170, 113, 115, 116, 114, 20]. Software defect density prediction mainly predicts the defect density or corresponding influencing factors of software modules. The relevant

research in this field is the least [98, 99]. The software defect tendency prediction mainly uses machine learning methods to construct classification model and then judge whether the new software modules contain defects or not. This kind of work is the focus of the current research [38, 17, 124, 132]. SDP in this thesis refers to software defect tendency prediction.

According to whether using labeled software modules and the corresponding amount of labeled modules used, SDP can be divided into supervised defect prediction, semi-supervised defect prediction and unsupervised defect prediction. More specifically, supervised defect prediction requires sufficient and labeled software modules as training sets to train a classification model, and then the model is used to predict whether the unlabeled modules contains defects or not. This is the most studied defect prediction task at present. The semi-supervised defect prediction assumes that only a small amount of labeled software modules is used as the training set. However, the limited labeled data can not accurately reflect the overall distribution of defect data, and can not train an effective and discriminant classification model. Since the large number of unlabeled software modules can describe the data distribution to a certain degree, thus some unlabeled modules can be selected to add into the training set to expand the scale [82, 123, 174]. At present, there is few work in this topic. Unsupervised defect prediction directly analyzes the unlabeled software modules and assigns them defect labels without any involvement of labeled modules.

For unsupervised defect prediction, according to different processing methods towards the software modules, it can be further divided into clustering based unsupervised defect prediction [105, 177, 11, 22, 173, 101, 160] and ranking based unsupervised defect prediction [166, 31, 51, 159, 52]. More specifically, clustering based unsupervised defect prediction uses clustering algorithm to group the unlabeled software modules into several clusters (generally two clusters, i.e., the defective group and the non-defective group),

and then assigns the defect label to each cluster based on certain rules. The labels of the software modules in one cluster are consistent with the label of the cluster [173]. Ranking based unsupervised defect prediction uses the size of module feature value to sort the modules, and then sets a threshold to label the modules. To be specific, the modules before and after this threshold are assigned different defect labels [178]. Since unsupervised defect prediction does not require the participation of labeled modules, it usually cannot achieve promising performance. This leads to that the relevant researches are far less concerned than supervised defect prediction.

For the supervised defect prediction, according to the different sources of the labeled data, it can be further divided into three different scenarios: **I**nner **V**ersion **D**efect **P**rediction (**IVDP**), **C**ross **V**ersion **D**efect **P**rediction (**CVDP**), and **C**ross **P**roject **D**efect **P**rediction (**CPDP**). The training data in IVDP scenario come from the labeled software modules of the same version of one project. The training set are usually generated using the cross validation method in experimental setting [138]. The training data in CVDP scenario come from labeled software modules of the previous versions of one project. The training data in CPDP scenario come from the labeled software modules of other external projects.

For CPDP, according to whether the feature sets of defect data across projects are the same or not, it can be further divided into homogeneous CPDP and heterogeneous CPDP. Homogeneous CPDP refers to that the feature sets of defect data from two projects are completely identical [86, 102, 152] or partially identical [142]. The latter one can use the common feature set of defect data from two projects to carry out CPDP task. Heterogeneous CPDP refers to that the feature sets of defect data from two projects are completely different [58, 100, 76, 75, 74, 140]. At present, homogeneous CPDP is the hot topic. The CPDP mentioned in this thesis refers to the scenario where the feature sets of cross project data are exactly the same. Figure 1.1 summarizes the structure diagram of

the current SDP research.

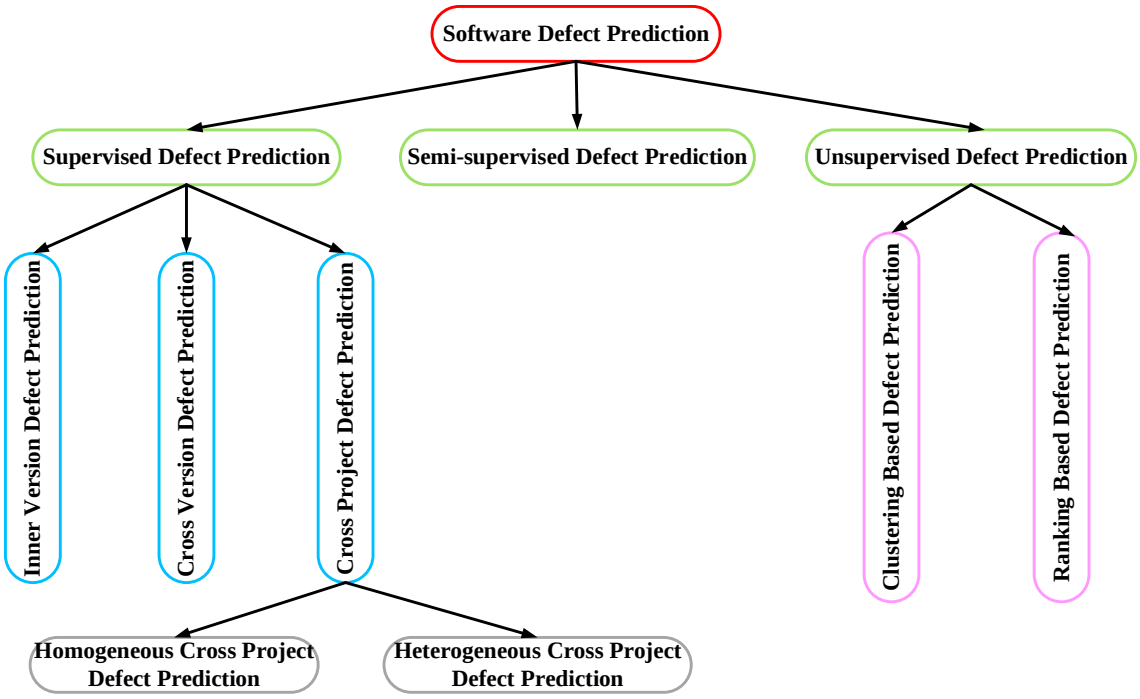


Figure 1.1: Research category for software defect prediction.

1.2 A Brief Introduction of Three Studied Defect Prediction Scenarios

In this thesis, we study machine learning assisted SDP under three defect prediction scenarios, i.e., inner version defect prediction, cross version defect prediction, and cross project defect prediction. The similarity of these three defect prediction scenarios lies in that they all belong to the supervised defect prediction, that is, labeled software modules are required as training sets to construct the classification model in advance. The difference lies in that the sources of labeled software modules as training sets are different under the three scenarios. Therefore, in different scenarios, the relationship between training set modules and test set modules is different, which leads to that the machine learning

methods used to solve the difficulties in the three scenarios vary. We briefly describe the basic content of each defect prediction scenario as follows:

(1) Inner Version Defect Prediction (IVDP)

IVDP is the most studied case among the three scenarios. For this prediction scenario, researchers mainly carried out the work from the following three perspectives: applying different feature engineering methods, classification models, and class imbalanced learning methods. The feature engineering method selects an optimal subset of features or learns new features to build the prediction model, rather than using all or original features. The reason is that some original features may be useless, redundant or cannot well reveal the structural information behind the data [155]. The commonly used feature engineering methods include chi-square [33, 158], information gain [127] and principal component analysis [96, 25]. The researches on classification models mainly explore the impacts of different classifiers on defect prediction performance [68, 34]. Most studies used classical classification models, such as random forest [37, 91, 163, 137], logistic regression [137, 164, 165, 167], and Naive Bayes [137, 148, 141]. Few studies explored and used novel classification models. The class imbalanced learning methods are mainly used to alleviate the negative effect of the imbalance of the defect data for the prediction performance. The widely used imbalanced learning methods include sampling based methods [8, 136], ensemble learning methods [67, 108], and cost-sensitive learning based methods [57, 80, 131].

(2) Cross Version Defect Prediction (CVDP)

CVDP is the least studied case among the three scenarios. For this prediction scenario, existing studies mainly used all software modules of the previous version to train the classification model, then the model was used to predict the labels of software modules in the current version [9]. CVDP is closer to the deployment in the real application scenario,

which uses the historical version data of the project to help identify the defect tendency of software modules in the current version under development [83].

(3) Cross Project Defect Prediction (CPDP)

CPDP is the second most studied defect prediction scenario. For the new software projects without the historical development data to collect the labels, CPDP uses the labeled data of other projects to conduct the prediction task for the unlabeled software modules of the new projects. Since the distribution of defect data in different projects generally has great dissimilarity, researchers have proposed two types of methods to solve this problem, i.e., the training data filtering methods [142] and the transfer learning methods [86]. Recently, researcher conducted an empirical analysis towards existing CPDP studies and pointed out that there was still much improvement room for the CPDP performance [43].

1.3 Challenges Faced in Each Defect Prediction Scenario

Due to the different sources of training sets under different defect prediction scenarios, we should treat the datasets from different angles. Thus, the problems to be solved and the used machine learning methods under different scenarios are different. The main difficulties to be solved under each defect prediction scenario are described as follows:

(1) Study on the feature learning and class imbalance issue for IVDP

The initial features of software defect data generally represent the basic characteristics of the code, such as code complexity and the changing characteristics. These initial features are often unable to well reveal the intrinsic structure information hidden behind the data [164, 149], that is, they may make it difficult to linearly distinguish the modules

of different labels. This will seriously reduce the performance of the classification model. In addition, class imbalance is prevalent in defect data in which the non-defective modules usually outnumber the defective ones. Class imbalance issue will degrade classifier performance because it makes most classifiers tend to predict the minority samples (i.e., the defective modules) as the majority samples (i.e., the non-defective modules). Thus, how to learn suitable feature representation to characterize the defect data and reduce the negative impacts of class imbalance for performance improvement are the main challenges in IVDP scenario.

(2) Study on the training data selection issue for CVDP

As the software development is an evolving process and software functions are increasingly complicated, the software modules undergo frequent changes during the version update. For example, the current version of the project inherits, refactories and deletes some existing modules from the prior version or adds some new modules. Such changes make that the modules from the prior version may not well represent the modules from the current version. These unrepresentative software modules will negatively affect the performance of the classification model for the CVDP task [153]. Thus, how to select some representative modules from the prior version to build more effective and targeted classification models for the performance improvement is the main challenge in CVDP scenario.

(3) Study on the data distribution similarity issue for CPDP

For the defect data of different projects, due to the different programming styles of developers, the different development platform environment, and the different functions and complexities contained in the projects, these will lead to that the data distributions of different projects vary. In order to build an effective CPDP model, the key is to reduce the data distribution differences across projects. The data distribution generally

includes two types, i.e., the marginal distribution and the conditional distribution. The marginal distribution refers to the distribution of module features themselves, while the conditional distribution refers to the distribution of module labels with known feature values. The similarity degree of two project data impacts the importance degree of the marginal distribution and conditional distribution. More specifically, when the data of two projects are much more dissimilar, the importance of the marginal distribution is higher than that of the conditional distribution, whereas when the data of two projects are similar, the conditional distribution is more important than the marginal distribution [145]. The transfer learning based methods are the mainstreams to solve the CPDP problem, such as the improved transfer component analysis [102]. However, existing CPDP methods based on transfer learning methods mainly reduce the marginal distribution differences of the defect data across projects without considering their conditional distribution differences. This greatly limits the performance of the transfer learning methods for the CPDP task. How to consider these two kinds of distribution differences simultaneously to adapt to different cross project data for the performance improvement is the main challenge in CPDP scenario.

1.4 Research Methods used in Each Defect Prediction Scenario

To address the above difficulties under different defect prediction scenarios, we propose different machine learning methods to solve them. The machine learning methods used in each defect prediction scenario are briefly described as follows:

(1) Dealing with feature representation and class imbalance for IVDP

The purpose of feature representation is to learn more discriminative features from the

original feature set for better representing the original data by mapping the features into a linearly separable space. The features in the new space help to improve the performance of the classification model. In the field of machine learning, the kernel method works well in feature representation learning. Thus, this thesis proposes a IVDP framework that applies the kernel based feature extraction method to learn more effective feature representation for the raw defect data.

The purpose of class imbalanced learning is to use some strategies to eliminate the bias caused by the difference in the number of modules with different labels to the decision of the classification model. It helps the model to identify defective software modules belonging to the minority classes. Since the sampling based imbalanced learning methods need change the data distribution and are not conducive to the model interpretation, and the ensemble based imbalanced learning is sensitive to outliers, the proposed IVDP framework uses a weighted classifier to alleviate the negative impact of the class imbalance.

(2) Dealing with training subset selection for CVDP

The purpose of training subset selection is to select an optimal module subset from the data of the previous version to replace the original ones for the classification task. The selected module subset can well represent the software modules in the current version. The existing studies for CVDP mainly use all software modules in the previous version to build the classification model, lacking related applications of training subset selection methods in the CVDP scenario. This thesis proposes a two-stage training subset selection framework based on the optimization solution to address this issue. First, the proposed CVDP framework selects a simplified subset of software modules from the labeled data of the previous version, and then picks up a module subset that is representative to the current version as the candidate training set. The classification model constructed on the candidate set is more targeted to the data of the current version and adapts to the classification task

for the data of the current version.

(3) Dealing with data distribution differences for CPDP

The purpose of distributed difference processing is to reduce the distribution differences between the defect data of different projects, making that the classification model trained on the processed data of one project can better fit the processed data of the other project. Since different cross project data pay different attentions to the marginal distribution difference and the conditional distribution difference, this thesis proposes a CPDP framework by introducing a novel transfer learning model. The model considers both the marginal distribution and conditional distribution, and minimizes the difference between the two distributions across the transformed cross project data. In addition, the model assigns different weights to the two distribution differences to better adapt to different cross project data.

1.5 Thesis Organization

The thesis is organized as follows:

Chapter 1 introduces some background knowledge, including the harm of software defects, the basic contents of software testing and software defect prediction. In addition, this chapter briefly introduces the three defect prediction scenarios used in this thesis, including the inner version defect prediction, cross version defect prediction, and cross project defect prediction, and then analyzes the difficulties faced under various software defect prediction scenarios. Finally, this chapter gives the corresponding machine learning methods to solve the difficulties.

Chapter 2 presents a review of some previous studies related to the work in this thesis,

including the research progress of the related work under the three defect prediction scenarios and the different machine learning assisted defect prediction techniques, such as the feature engineering methods, class imbalanced learning methods, training subset selection methods, and transfer learning methods for the defect prediction task.

Chapter 3 proposes a composite framework that combines a kernel function based feature extraction and a state-of-the-art weighted version classifier to solve the problem of feature learning and class imbalance issues respectively in the scenario of inner version defect prediction (IVDP).

Chapter 4 proposes a two-stage instance selection framework based on optimization method to address the training subset selection issue in the scenario of cross version defect prediction (CVDP).

Chapter 5 introduces a transfer learning model which considers the marginal and conditional distribution differences as well as their different weights to deal with the data distribution difference issus across projects in the scenario of cross project defect prediction (CPDP).

Chapter 6 summarizes the research contents under the three defect prediction scenarios (i.e., IVDP, CVDP, and CPDP) in this thesis, and gives some unsolved or unexplored problems that will be focused in the future.

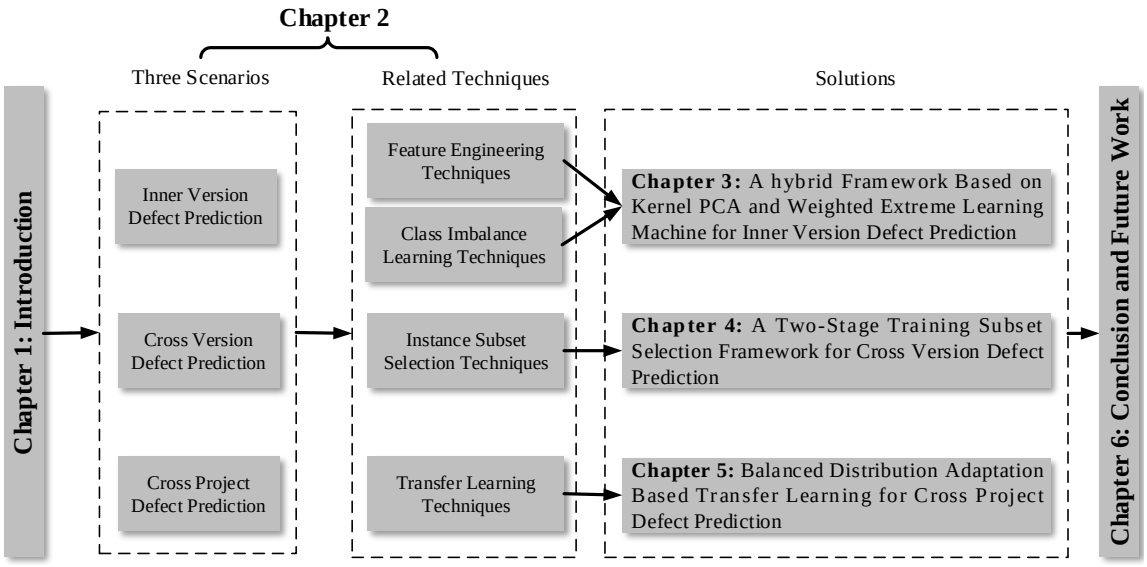


Figure 1.2: The organization chart of this thesis.

Chapter 2

Literature Review

2.1 Studies on Defect Prediction Under Different Scenarios

SDP builds models to predict software modules with possible defect risk by mining and analyzing the rich historical development data in the software repository, so as to optimize the strategy of test resource allocation, reduce the test cost, and improve the reliability and security of the software. In general, SDP methods use labeled software defect data to build statistical or machine learning models. Software defect data includes software module features and label information. More specifically, module features include code complexity features, code change features, network features of function call relationships, and so on. The label information is usually a binary variable used to indicate whether a software module contains defects or not. Whether the defective modules can be identified using the historical development data is an important criteria to evaluate the performance of the constructed defect prediction models.

Most of the existing researches regard defect prediction as a supervised learning task, that is, the labeled defect data are first used as training set to construct a classification

model, and then the model is used to predict whether the unlabeled modules contain defects or not. As mentioned in Chapter 1, according to the source of this training set, the existing studies can be divided into three different defect prediction scenarios, including IVDP, CVDP, and CPDP.

2.1.1 Studies on IVDP

For IVDP, the labeled software module data used as the training set come from the historical data of the current version of the same project, and the remaining data of the project version are used to evaluate the performance of the built classification model. This scenarios is the most popular branch of research. Researchers have proposed various methods, such as machine learning methods, to solve related problems. In addition, different classification models were applied to the IVDP. This subsection mainly discusses the existing research work on analyzing the impact of different classification models on IVDP performance.

Various classification models have been applied to defect prediction. Malhotra [88] evaluated the feasibility of seven classification models for defect prediction by conducting a systematic literature review on the studies that published from January 1991 to October 2013. They discussed the merits and demerits of the classification models and found that they were superior to traditional statistical models. In addition, they suggested that new methods should be developed to further improve the defect prediction performance. Malhotra [89] used the statistical tests to compare the performance differences among 18 classification models for defect prediction. They performed the experiments on seven Android software projects and stated that these models have significant differences while support vector machine and voted perceptron model did not perform well. Lessmann et al. [68] conducted an empirical study to investigate the effectiveness of 21 classifiers

on NASA dataset. The results showed that the performances of most classifiers have no significant differences. They suggested that some additional factors, such as the computational overhead and simplicity, should be considered when selecting a proper classifier for defect prediction. Ghotra et al. [34] expanded Lessmann's experiment by applying 31 classifiers to two versions of NASA dataset and PROMISE dataset. The results showed that these classifiers achieved similar results on the noisy NASA dataset but different performance on the clean NASA and the PROMISE datasets. Malhotra et al. [90] investigated the performances of 18 classifiers on six projects with object-oriented features and found that Naive Bayes classifier achieved the best performance. Some researchers introduced KPCA into defect prediction [117, 84, 85] recently, and they aimed at building asymmetrical prediction models with the kernel method by considering the relationship between principal components and the class labels. In this thesis, we leverage KPCA as a feature selection method to extract representative features for defect prediction. In addition, Mesquita et al. [94] proposed a method based on **E**xtreme **L**earning **M**achine (**ELM**) with reject option (i.e., IrejoELM) for defect prediction. The results were good because they abandoned the modules that have contradictory decisions for two designed classifiers. However, in practice, such modules should be considered.

2.1.2 Studies on CVDP

For CVDP, the labeled software module data used as the training set come from the historical data of the previous version of the same project, and data of current version are used to evaluate the performance of the built classification model. From the fact that the training set data also come from the same project, both CVDP and IVDP can belong to the within project defect prediction. Compared with IVDP, CVDP is closer to the deployment in a real-world application scenario in which the historical version data of the project is used to assist in the identification of the defect-prone modules in its on-going version.

However, to the best of our knowledge, only a fewer related studies have been devoted to CVDP.

Bennin et al. [9] evaluated the CVDP performance of 11 classification models on 25 open source software projects (each with two versions) with an effort-aware indicator. The experimental results showed that M5 and K* models achieved the best performance but were also greatly affected by the defect ratio and size of the defect data. However, the performance differences among all 11 methods were not statistically significant. Holschuh et al. [45] explored the CVDP performance on a large software system by collecting four types of features. The experiments on six projects (each with three versions) showed that the overall performance is unsatisfactory. Shukla et al. [130] treated CVDP as a multi-objective optimization problem with two objective functions: i.e., maximizing recall by minimizing misclassification cost and the cost of quality assurance activities on defect prone modules. They used four traditional classification models to conduct experiments on 11 open source projects with total 30 cross-version pairs and found that multi-objective logistic regression outperformed four single-objective algorithms. Li et al. [73] compared the CVDP performance of a multi-objective approach which optimized two objectives and a single-objective approach which optimized the trade-off of the two objectives. They conducted experiments on four open source software projects with a total of 10 cross-version pairs. The results indicated that when the trade-off was known, single-objective approach can achieve better performance which was not consistent with the conclusion in [130]. Yang et al. [162] investigated the CVDP performance of two models, i.e., ridge regression and lasso regression. They conducted experiments on 11 projects with a total of 30 cross-version pairs and found that the two methods achieved better performance compared with four baseline methods. But this work focused on the ranking task, not the classification task as we do.

Although some previous studies [95, 64, 64, 168, 147] also mentioned the CVDP

concept, they did not specialize in CVDP, just treated it as one of multiple defect prediction scenarios, like IVDP and CPDP. Thus, we do not discuss them here. All these mentioned studies fed all modules of the prior version into the classification model without taking the distribution differences caused by the representation of modules into account and conducted a small scale experiments on a few project versions. Different from these studies, this thesis focuses on selecting an optimal training modules to relieve the gap of distribution differences across versions and conducts a large scale experiments on total 50 cross-version pairs for CVDP task.

Recently, there are two studies that have considered the issue of distribution differences for CVDP task. Lu et al. [83] selected some unlabeled modules as candidate from the current version with an active learning method and labeled these modules by querying the software experts, then added them into the prior version to form a mixed training set. They expected that these modules could alleviate the distribution differences by supplementing some distribution information of the current version into the prior version. The experimental results on three Eclipse projects with a total of six cross-version pairs showed that their method could improve the CVDP performance. However, the candidate modules selected by the active learning method were only informative while not representative for the data of the current version [72, 53]. Inspected by Lu et al.'s work, Xu et al. [153] proposed a hybrid active learning method to select both informative and representative modules from the current version and merged them into the prior version to construct an enhanced training set. They conducted experiments on 10 open source projects with a total of 31 cross-version pairs and found that their method could further improve the CVDP performance compared with Lu et al.'s method. Both studies selected some modules from the current version and added them into the prior version. However, their methods needed to label the modules selected from the current version which involves in additional efforts.

2.1.3 Studies on CPDP

For CPDP, the labeled software module data used as the training set come from other projects (also called source projects), and data from current project (also called target project) are used to evaluate the performance of the built classification model. There are two kinds of CPDP scenarios: one is that the feature sets of source project and target project are the same of have a common part, also known as homogeneous CPDP and the other is that the feature sets of source project and target project are different, also known as heterogeneous CPDP. Since the homogeneous CPDP is the research hotspot, we focus on this topic and introduce some related studies in this subsection.

To the best of our knowledge, Briand et al. [14] were the first to explore whether the CPDP model built on one system for another system was worth investigating. However, the experimental results on two java systems implied that such a model achieved poor performance. Another early study about CPDP is performed by Zimmermann et al. [179]. The experimental results on a total of 622 cross-project pairs with logistic regression classifier showed that only 3.4% pairs achieved satisfactory performances. The reason of the disappointing results from these early studies is that they conducted CPDP by using all modules of the source project to train the classification model without considering the data distribution differences of the two projects. To address this issue, recently, researchers have proposed different methods to narrow the gap of the distribution differences between the cross project data. Existing related studies can be roughly divided into two groups: the training data filter based CPDP methods and transfer learning based CPDP methods. The related work about the two methods are described in Subsection 2.2.3 and Subsection 2.2.4. In this subsection, we mainly introduce some comprehensively empirical studies about CPDP.

Hosseini et al. [46] made a systematic literature review of the CPDP studies and selected

part of work for detailed analysis. They identified the most commonly used performance indicators, well-performing classification models, and widely used data sets. They pointed out that CPDP remains a challenging task that deserves more attentions. To determine which CPDP method performed best, Herbold et al. [42] reproduced some existing CPDP methods and evaluated their performance on five datasets. The experimental results showed that three methods could achieve the best performance in most cases, and suggested that there was still room for improvement if the CPDP methods were put into practical applications. Similarly, Porto et al. [109] replicated several existing CPDP methods and compared them on 47 versions of 15 projects in the PROMISE dataset. They identified four methods which could achieve the best performance and proposed a cross project model with meta-learning.

2.2 Studies on Different Methods for Defect Prediction

In this section, we mainly introduce the related work of some machine learning methods used under the above three defect prediction scenarios.

2.2.1 Feature Selection for Defect Prediction

Feature selection methods simplify the defect data by selecting part of representative features from the original feature set. The commonly used methods include filter-based feature ranking methods and wrapper-based feature subset selection methods. The former one ranks the features according to their importance towards the labels and selects a certain number of top ranked features. The latter one selects a feature subset that can achieve the best result for a given classification model and performance measurement.

Some recent studies explored the effectiveness of feature selection methods for defect

prediction performance. Song et al. [133] suggested that feature selection is an indispensable part of a general defect prediction framework. Menzies et al. [93] found that naive Bayes classifier with information gain based feature selection can get good performances over 10 projects from the NASA dataset. Shivaji et al. [128, 129] studied the performance of filter-based and wrapper-based feature selection methods for bug prediction. Their experiments showed that feature selection can improve the defect prediction performance even remaining 10% of the original features. Gao et al. [151] investigated four filter-based feature selection methods on a large telecommunication system and found that the Kolmogorov-Smirnov method achieved the best performance. Gao et al. [33] explored the performance of their hybrid feature selection framework based on seven filter-based and three feature subset search methods. They found that the reduced features would not adversely affect the prediction performance in most cases. Chen et al. [21] modelled the feature selection as a multi-objective optimization problem: minimizing the number of selected features and maximizing the defect prediction performance. They conducted experiments on 10 projects from PROMISE dataset and found that their method outperformed three wrapper-based feature selection methods. However, their method was less efficient than two wrapper-based methods. Catal et al. [16] conducted an empirical study to investigate the impact of the dataset size, the types of feature sets and the feature selection methods on defect prediction. To study the impact of feature selection methods, they first utilized a **C**orrelation-based **F**eature **S**election (**CFS**) method to obtain the relevant features before training the classification models. The experiments on five projects from NASA dataset showed that the random forest classifier with CFS performed well on large project datasets and the Naive Bayes classifier with CFS worked well on small projects datasets. Xu et al. [154] conducted an extensive empirical comparison to investigate the impact of 32 feature selection methods on defect prediction performance over three public defect datasets. The experimental results showed that the performance of these methods had significant differences on all datasets and that PCA performed the

worst. Ghotra et al. [35] extended Xu et al.'s work and conducted a large-scale empirical study to investigate the defect prediction performance of 30 feature selection methods with 21 classification models. The experimental results on 18 projects from NASA and PROMISE datasets suggested that CFS method with best-first search strategy achieved the best performance among all other feature selection methods on most projects.

2.2.2 Class Imbalanced Learning for Defect Prediction

Since class imbalance issue can hinder defect prediction techniques to achieve satisfactory performance, researchers have proposed different imbalanced learning methods to mitigate such negative effects. Sampling based methods, ensemble based methods, and cost-sensitive based methods are the most studied imbalanced learning methods for defect prediction.

For the sampling based imbalanced learning methods, there are two main sampling strategies to balance the data distribution. One is to decrease the number of non-defective modules (such as under-sampling technique), the other is to increase the number of the defective modules with redundant modules (such as over-sampling technique) or synthetic modules (such as **S**ynthetic **M**inority **O**ver-sampling **TE**chnique, SMOTE). Kamei et al. [60] investigated the impact of four sampling methods on the performance of four basic classification models. They conducted experiments on two industry legacy software systems and found that these sampling methods can benefit linear and logistic models but were not helpful to neural network and classification tree models. Bennin et al. [7] assessed the statistical and practical significance of six sampling methods on the performance of five basic defect prediction models. Experiments on 10 projects indicated that these sampling methods had statistical and practical effects in terms of some performance indicators, such as Pd, Pf, G-mean, but had no effect in terms of AUC. Bennin

et al. [6] explored the impact of a configurable parameter (i.e, the percentage of defective modules) in seven sampling methods on the performance of five classification models. The experimental results showed that this parameter can largely impact the performance (except AUC) of studied prediction models. Due to the contradictory conclusions of previous empirical studies about which imbalanced learning methods performed the best in the context of defect prediction models, Tantithamthavorn et al. [136] conducted a large-scale empirical experiment on 101 project versions to investigate the impact of four popularly-used sampling techniques on the performance and interpretation of seven classification models. The experimental results explained that these sampling methods increased the completeness of recall indicator but had no impact on the AUC indicator. In addition, the sampling based imbalanced learning methods were not conducive to the understanding towards the interpretation of the defect prediction models.

Ensemble based imbalanced learning methods usually combine with multiple weak classifiers to improve the overall defect prediction performance [143]. Wang et al. [146] compared sampling based, threshold moving based and ensemble learning methods. They found that a variant of AdaBoost achieved the best overall performance. Sun et al. [135] proposed a coding-based ensemble learning method which converted the imbalanced data with two labels into multiple labels and then used a special coding strategy to build a defect prediction model. Laradji et al. [67] combined the ensemble learning method and feature selection method to alleviate class imbalance issue of defect data. Petric et al. [108] first constructed an ensemble model with a weighted diversity technique to explore the diversity of four classifiers, and then combined the classifiers using a stacking technique. The experimental results on eight projects showed that the proposed method could achieve good performance. Yang et al. [163] proposed a two-layer ensemble learning method for just-in-time defect prediction. In the inner layer, the random forest models were constructed by combining decision tree and bagging methods. In the outer layer, the

random undersampling technique was used to train various random forest models, and the stacking technique was used to combine the models. Ensemble learning methods usually requires iteratively sampling multiple module subsets to improve the performance.

The cost-sensitive based imbalanced learning methods alleviate the differences between the instance number of two classes by assigning different weights to the two types of instances. Khoshgottar et al. [63] proposed a cost-boosting method by combining multiple classification models. Experiments on two industrial software systems showed that the boosting method was feasible for defect prediction. Zheng et al. [175] proposed three cost-sensitive boosting methods to boost neural networks for defect prediction. Experimental results showed that threshold-moving-based boosting neural networks can achieve better performance, especially for object-oriented software projects. Liu et al. [80] proposed a novel two-stage cost-sensitive learning method by utilizing cost information in the classification stage and the feature selection stage. Experiments on seven projects of NASA dataset demonstrated its superiority compared with the single-stage cost-sensitive classifiers and cost-blind feature selection methods. Siers et al. [131] proposed two cost-sensitive classification models by combining decision trees to minimize the classification cost for defect prediction. The experimental results on six projects of NASA dataset showed the superiority of their methods compared with six classification methods. The WELM technique used in this thesis for the IVDP task belongs to this type of imbalanced learning methods.

2.2.3 Training subset selection for Defect Prediction

Training subset selection methods use some rules to select part of software modules from the training set in which the reserved modules are important to the modules of the test set.

Turhan et al. [142] proposed a nearest neighbor filter method to select a module subset

from the source project. More specifically, for each module in the target project, this method first selected its top-k nearest modules, then these candidate modules without duplication constituted the training set. Peter et al. [107] proposed a subset selection strategy with a clustering algorithm. More concretely, this method first combined the data of the two projects, then used k-means clustering algorithm to cluster the mixed data and discarded the clusters that did not contain any module of the target project. For each module of the source project in the remaining clusters, the method found its nearest neighbor module in the target project (called popular modules). Then, for each popular module, this method selected its greatest fan (the nearest module in the source project). Finally, these selected fans were fed into the classification model. The results showed that this method led to the performance improvement compared with the method in [142] and the IVDP performance. Kawata et al. [62] proposed a relevancy filter method for source project data simplification. This method first used the DBSCAN algorithm to cluster the mixed project data. For the clusters that contained at least one module of the target project, the modules of the source project in such clusters formed the final training data. Following Kawata et al.'s work, Yu et al. [171] proposed a new subset selection method by just replacing the DBSCAN clustering algorithm with agglomerative clustering. The experimental results showed that this method achieved a small improvement compared with the method in [62]. Ryu et al. [119] proposed a hybrid instance selection method based on nearest neighbor. This method learns the local knowledge using k neighbor algorithm and learns the global knowledge using Bayesian learning, and then comprehensively selects the appropriate module subset. The experimental results on seven projects from the NASA dataset showed that this method could obtain higher detection rate and lower rate of false positives. Herbold et al. [41] proposed two training data selection methods based on distance measurement in which one method is based on EM clustering algorithm and the other one is based on the nearest neighbor algorithm. Their experimental results on 71 project versions from the PROMISE

dataset showed that their method improved CPDP performance, but achieved worse IVDP performance. He et al. [39] proposed a similarity based training data selection method which ranked all modules in the training set based on an improved scoring strategy. The experimental results on 10 projects from the PROMISE dataset and five projects from the AEEEM dataset demonstrated the effectiveness of the proposed method. In addition, the results showed that the combination of Euclidean distance and linear normalization could obtain the best performance.

2.2.4 Transfer Learning for Defect Prediction

Transfer learning based methods are mainly applied to CPDP scenario. This kind of methods transform the source and target project data into a common feature space, aiming to minimize the data distribution differences among the two project data. Thus, the classification model built on the mapped source project data is more effective than that built on the original source project data to predict the labels of the target project data.

To the best of our knowledge, Ma et al. [86] were among the first to introduce transfer learning into CPDP. Instead of discarding some modules of the source project, they proposed a transfer naive Bayes (TNB) method to transfer the valuable information of the source project into the target project. TNB first utilizes the data gravitation formula to measure the similarity of the modules of the source project to the modules of the target project, and assigns different weights to the modules of the source project based on the similarity, then integrates the weight information into the Bayes formula to develop a weighted Naive Bayes method based transfer learning. The experiments on seven defect data from NASA dataset and three defect data from SOFTLAB dataset showed that TNB achieved better performances than the method in [142]. Nam et al. [102] proposed an extended transfer component analysis (TCA) method, called TCA+, to learn some transfer

components for cross project data in a kernel Hilbert space. TCA+ first defines some rules to find the optimum data normalization strategy, and then applies the original TCA method to make the data distributions of the two projects closer. The experiments on five defect data from AEEEM dataset and three defect data from RELINK dataset showed that TCA+ achieved competitive performance compared with the IVDP setting and the original TCA method. Chen et al. [19] proposed a transfer learning method, called double transfer boosting (DTB), for CPDP. DTB first re-weights the modules of the source project based on data gravitation formula and then applies a transfer boosting method to eliminate some negative modules from the source project. The experiments showed that DTB outperformed four baseline CPDP methods and achieved better performances than three IVDP methods. The main drawback of DTB is that the used transfer boosting method needs the participation of some labeled modules from the target project, which limits its usage under the scenario in which the target project has no labeled modules. Ryu et al. [120] proposed a transfer cost-sensitive boosting (TCSBoost) method that considers both knowledge transfer and class imbalance for CPDP. TCSBoost first calculates the similarity weight between the source and the target projects, and employs a resampling method to rebalance the data distribution of the defective and non-defective classes of the source project, then applies a cost-sensitive boost method to deal with the distribution differences between the two project data. Like the DTB method, TCSBoost requires a small amount of labeled modules of the target project, which hinders its usage in the general CPDP scenario. Liu et al. [78] proposed a two-phase transfer learning (TPTL) model. In the first stage, TPTL selects two source projects, having the highest distribution similarity to the target project and the best performance, as candidates from a set of source projects. In the second stage, TPTL utilizes the TCA+ method to build two transfer learning models based on the two candidates to conduct CPDP. The focus of their work is on the selection of the candidate source projects. Different from the above transfer learning methods that usually only consider the marginal distribution difference of the cross project data, in this thesis,

we introduce a state-of-the-art transfer learning method which considers both marginal and conditional distribution differences as well as their weights for the CPDP task.

Chapter 3

A hybrid Framework Based on Kernel PCA and Weighted Extreme Learning Machine for Inner Version Defect Prediction

Software testing is an important part of software development life cycle for software quality assurance [139, 97]. Defect prediction can assist the quality assurance teams to reasonably allocate the limited testing resources by detecting the potentially defective software modules (such as classes, files, components) before releasing the software product. Thus, effective defect prediction can save testing cost and improve software quality [124, 133, 161].

The majority of existing researches leverages various machine learning techniques to build defect prediction methods. In particular, many classification techniques have been used as defect prediction models, such as decision tree, Naive Bayes, random forest, nearest neighbor, and logistic regression. Since irrelevant and redundant features in the defect data may degrade the performance of the classification models, different feature selection methods have been applied to select an optimal feature subset for defect

prediction [154, 35]. These methods can be roughly divided into three categories: the filter-based feature ranking methods, wrapper-based feature subset evaluation methods, and extraction-based feature transformation methods.

3.1 Motivation

Selecting optimal features that can reveal the intrinsic structures of the defect data is crucial to build effective defect prediction models. The filter-based and wrapper-based feature selection methods only select a subset of the original features without any transformation [134]. However, such raw features may not properly represent the essential structures of raw defect data [149]. Being a linear feature extraction method, PCA has been widely used to transform the raw features to a low-dimensional space where the features are the linear combinations of the raw ones [79, 15, 176, 65]. PCA performs well when the data are linearly separable and follow a Gaussian distribution, whereas the real defect data may have complex structures that can not be simplified in a linear subspace [125, 36]. Therefore, the features extracted by PCA are usually not representative, and cannot gain anticipated performance for defect prediction [92, 154]. To address this issue, we exploit KPCA [121], a nonlinear extension of PCA, to project the original data into a latent high-dimensional feature space in which the mapped features can properly characterize the complex data structures and increase the probability of linear separability of the data. When the original data follow an arbitrary distribution, the mapped data by KPCA obey an approximate Gaussian distribution. Figure 3.1 shows the merit of the feature mapping, where the data are linearly inseparable within the low-dimensional space but linearly separable within the high-dimensional space. Existing studies have shown that KPCA outperforms PCA [122, 66].

Although many classifiers have been used for defect prediction, Lessmann et al. [68]

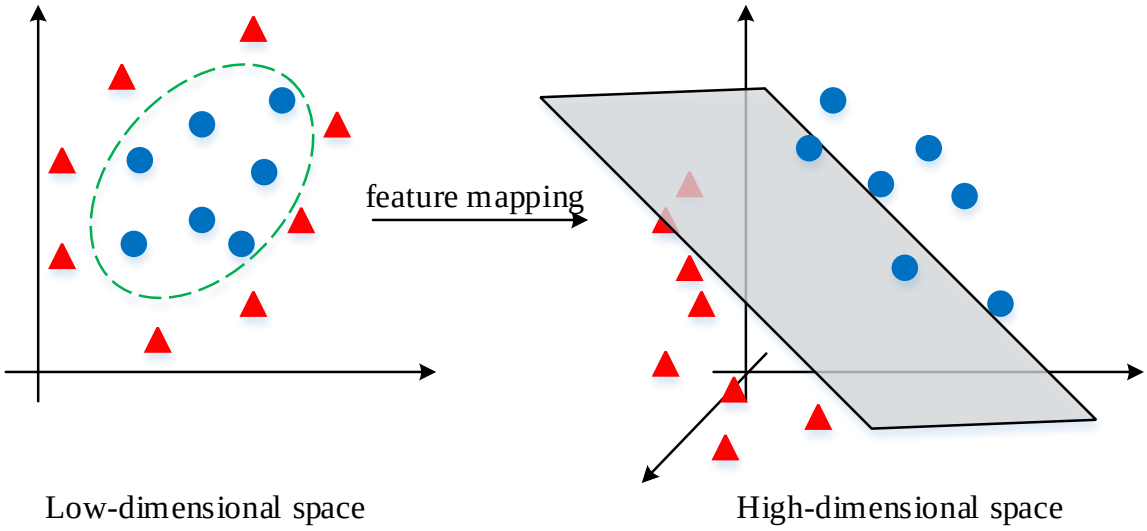


Figure 3.1: An example of the merit of feature mapping.

suggested that the selection of classifiers for defect prediction needs to consider additional criteria, such as computational efficiency and simplicity, because they found that there are no significant performance differences among most defect prediction classifiers. Moreover, class imbalance is prevalent in defect data in which the non-defective modules usually outnumber the defective ones. It makes most classifiers tend to classify the minority samples (i.e., the defective modules) as the majority samples (i.e., the non-defective modules). However, existing defect prediction methods did not address this problem well, thus leading to unsatisfactory performance. In this chapter, we exploit **S**ingle-hidden **L**ayer **F**eedforward **N**eural network**s** (**SLFNs**) called **W**eighted **E**xtreme **L**earning **M**achine (**WELM**) [180] to overcome this challenge. WELM assigns higher weights to defective modules to emphasize their importance. In addition, WELM is efficient and convenient since it only needs to adaptively set the number of hidden nodes while other parameters are randomly generated instead of being tuned through iterations like traditional neural networks [50].

In this chapter, we propose a new IVDP framework called **KPWE** that leverages the two aforementioned techniques: **KP**CA and **WE**LM. This framework consists of two major

stages. First, KPWE exploits KPCA to map original defect data into a latent feature space. The mapped features in the space can well represent the original ones. Second, with the mapped data, KPWE applies WELM to build an efficient and effective defect prediction model that can handle the class imbalance issue.

3.2 The Used Methods and Proposed IVDP Framework

To deal with the feature representation and class imbalance issue of software modules in the inner version scenario, we propose a new defect prediction framework KPWE. This framework consists of two stages: feature extraction and model construction. Subsection 3.2.1 describes how to project the original data into a latent feature space using the nonlinear feature transformation technique KPCA, subsection 3.2.2 introduces the basic ELM model, subsection 3.2.3 presents how to build the WELM based classification model with the extracted features by considering the class imbalance issue, subsection 3.2.4 gives our proposed IVDP framework KPWE.

3.2.1 Feature Extraction Based on KPCA

In this stage, we extract representative features with KPCA to reveal the potentially complex structures in the defect data. KPCA uses a nonlinear mapping function φ to project each raw data point within a low-dimensional space into a new point within a high-dimensional feature space F.

Given a dataset $\{x_i, y_i\}, i = 1, 2, \ldots, n$, where $x_i = [x_{i1}, x_{i2}, \ldots, x_{im}]^{\mathrm{T}} \in \mathbb{R}^m$ denotes the feature set and $y_i = [y_{i1}, y_{i2}, \ldots, y_{ic}]^{\mathrm{T}} \in \mathbb{R}^c$ ($c = 2$ in this work) denotes the label set.

Assuming that each data point x_i is mapped into a new point $\varphi(x_i)$ and the mapped data

points are centralized, i.e.,

$$\frac{1}{n}\sum_{i=1}^{n}\varphi(x_i)=0 \tag{3.1}$$

The covariance matrix $\mathbf{C}$ of the mapped data is:

$$\mathbf{C}=\frac{1}{n}\sum_{i=1}^{n}\varphi(x_i)\varphi(x_i)^{\mathrm{T}} \tag{3.2}$$

To perform the linear PCA in F, we diagonalize the covariance matrix C, which can be treated as a solution of the following eigenvalue problem

$$\mathbf{CV}=\lambda\mathbf{V}, \tag{3.3}$$

where λ and $\mathbf{V}$ denote the eigenvalues and eigenvectors of C, respectively.

Since all solutions $\mathbf{V}$ lie in the span of the mapped data points $\varphi(x_1),\varphi(x_2),\ldots,\varphi(x_n)$, we multiply both sides of Eq.(3.3) by $\varphi(x_l)^T$ as

$$\varphi(x_l)^{\mathrm{T}}\mathbf{CV}=\lambda\varphi(x_l)^{\mathrm{T}}\mathbf{V},\forall l=1,2,\ldots,n \tag{3.4}$$

Meanwhile, there exist coefficients $\alpha_1,\alpha_2,\ldots,\alpha_n$ that linearly express the eigenvectors $\mathbf{V}$ of $\mathbf{C}$ with $\varphi(x_1),\varphi(x_2),\ldots,\varphi(x_n)$, i.e.,

$$\mathbf{V}=\sum_{j=1}^{n}\alpha_j\varphi(x_j) \tag{3.5}$$

Eq.(3.4) can be rewritten as following formula by substituting Eq.(3.2) and Eq.(3.5) into it

$$\frac{1}{n}\varphi(x_l)^{\mathrm{T}}\sum_{i=1}^{n}\varphi(x_i)\varphi(x_i)^{\mathrm{T}}\sum_{j=1}^{n}\alpha_j\varphi(x_j)=\lambda\varphi(x_l)^{\mathrm{T}}\sum_{j=1}^{n}\alpha_j\varphi(x_j) \tag{3.6}$$

Let the kernel function $\kappa(x_i,x_j)$ be

$$\kappa(x_i,x_j)=\varphi(x_i)^{\mathrm{T}}\varphi(x_j) \tag{3.7}$$

Then Eq.(3.6) is rewritten as

$$\frac{1}{n}\sum_{l=1,i=1}^{n}\kappa(x_l,x_i)\sum_{i=1,j=1}^{n}\alpha_j\kappa(x_i,x_j)=\lambda\sum_{l=1,j=1}^{n}\alpha_j\kappa(x_l,x_j) \tag{3.8}$$

Let the kernel matrix $\mathbf{K}$ with size $n\times n$ be

$$\mathbf{K}_{i,j}=\kappa(x_i,x_j) \tag{3.9}$$

Then Eq.(3.8) is rewritten as

$$\mathbf{K}^2\alpha=n\lambda\mathbf{K}\alpha, \tag{3.10}$$

where $\alpha=[\alpha_1,\alpha_2,\ldots,\alpha_n]^{\mathrm{T}}$.

The solution of Eq (3.10) can be obtained by solving the eigenvalue problem

$$\mathbf{K}\alpha=n\lambda\alpha \tag{3.11}$$

for nonzero eigenvalues λ and corresponding eigenvectors α. As we can see, all the solutions of Eq.(3.11) satisfy Eq.(3.10).

As mentioned above, we first assume that the mapped data points are centralized. If they are not centralized, the Gram matrix $\tilde{\mathbf{K}}$ be used to replace the kernel matrix $\mathbf{K}$ as

$$\tilde{\mathbf{K}}=\mathbf{K}-1_n\mathbf{K}-\mathbf{K}1_n+1_n\mathbf{K}1_n, \tag{3.12}$$

where 1_n denotes the $n\times n$ matrix with all values equal to $1/n$.

Thus, we just need to solve the following formula

$$\tilde{\mathbf{K}}\alpha=n\lambda\alpha \tag{3.13}$$

To extract the nonlinear principal components of a new test data point $\varphi(x_{new})$, we can compute the projection of the k-th kernel component by

$$\mathbf{V}^k\cdot\varphi(x_{new})=\sum_{i=1}^{n}\alpha_i^k\varphi(x_i)^{\mathrm{T}}\varphi(x_{new})=\sum_{i=1}^{n}\alpha_i^k\kappa(x_i,x_{new}) \tag{3.14}$$

Figure 3.2 depicts the process of KPCA for feature extraction. KPCA simplifies the feature mapping by calculating the inner product of two data points with kernel function instead of calculating the $\varphi(x_i)$ explicitly. Various kernel functions, such as Gaussian Radial Basic Function (RBF) kernel and polynomial kernel, can induce different nonlinear mapping. The RBF kernel is commonly used in image retrieval and pattern recognition domains [106, 70] that is defined as

$$\kappa(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \tag{3.15}$$

where $\|\cdot\|$ denotes the l_2 norm and $2\sigma^2 = \omega$ denotes the width of the Gaussian RBF function.

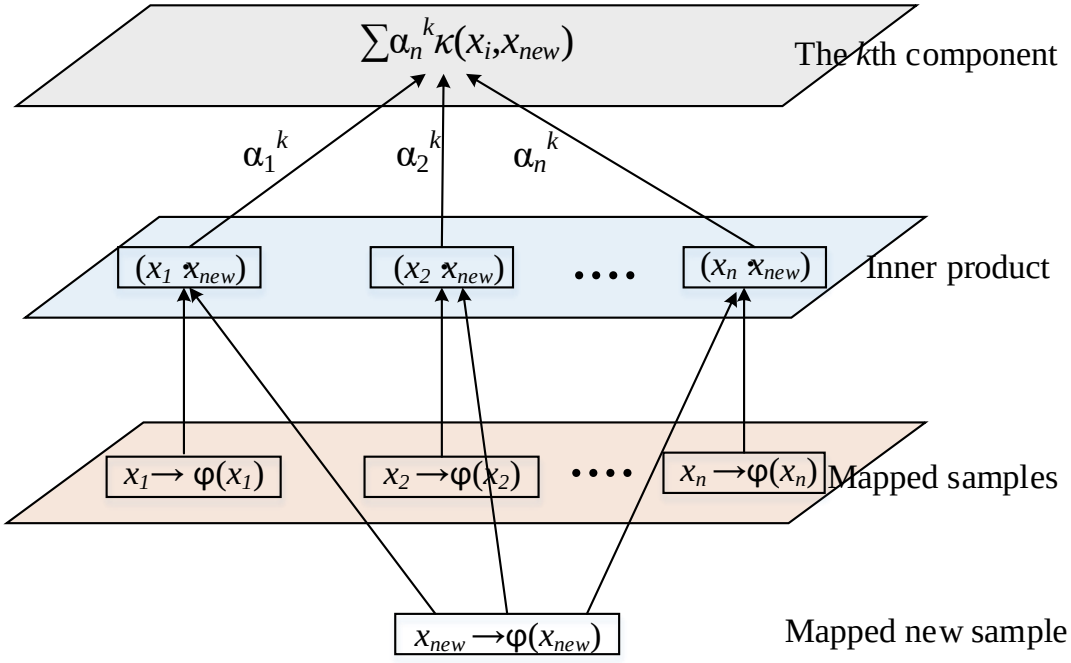


Figure 3.2: Feature extraction with KPCA.

To eliminate the underlying noise in the data, when performing the PCA in the latent feature space F, we maintain the most important principal components that capture at least 95% of total variances of the data according to their cumulative contribution rates [1]. Finally, the data are mapped into a p-dimensional space.

After completing feature extraction, the original training data are transformed to a new dataset $\{x'_i, y_i\} \in \mathbb{R}^p \times \mathbb{R}^c$ $(i = 1, 2, \ldots, n)$.

3.2.2 ELM

Before formulizing the WELM, we first introduce the basic ELM [50].

With the mapped dataset $\{x'_i, y_i\} \in \mathbb{R}^p \times \mathbb{R}^c$ $(i = 1, 2, \ldots, n)$, the output of the generalized SLFNs with q hidden nodes and activation function $\mathbf{h}(x')$ is formally expressed as

$$o_i = \sum_{k=1}^{q} \beta_k \mathbf{h_k}(x'_i) = \sum_{k=1}^{q} \beta_k h(w_k, b_k, x'_i), \tag{3.16}$$

where $i = 1, 2, \ldots, n$, $w_k = [w_{k1}, w_{k2}, \ldots, w_{kp}]^{\mathrm{T}}$ denotes the input weight vector connecting the input nodes and the k-th hidden node, b_k denotes the bias of the k-th hidden node, $\beta_k = [\beta_{k1}, \beta_{k2}, \ldots, \beta_{kc}]^{\mathrm{T}}$ denotes the output weight vector connecting the output nodes and the k-th hidden node, and o_i denotes the expected output of the i-th sample. The commonly-used activation functions in ELM include sigmoid function, Gaussian RBF function, hard limit function, and multiquadric function [48, 26]. Figure 3.3 depicts the basic architecture of ELM.

Eq.(3.16) can be equivalently rewritten as

$$\mathbf{H}\beta = \mathbf{O}, \tag{3.17}$$

where $\mathbf{H}$ is called the hidden layer output matrix of the SLFNs and is defined as

$$\begin{aligned} \mathbf{H} = \mathbf{H}(w_1, \ldots, w_q, b_1, \ldots, b_q, x'_1, \ldots, x'_n) &= \begin{bmatrix} \mathbf{h}(x'_1) \\ \vdots \\ \mathbf{h}(x'_n) \end{bmatrix} \\ &= \begin{bmatrix} h(w_1, b_1, x'_1) & \cdots & h(w_q, b_q, x'_1) \\ \vdots & \ddots & \vdots \\ h(w_1, b_1, x'_n) & \cdots & h(w_q, b_q, x'_n) \end{bmatrix}_{n \times q}, \end{aligned} \tag{3.18}$$

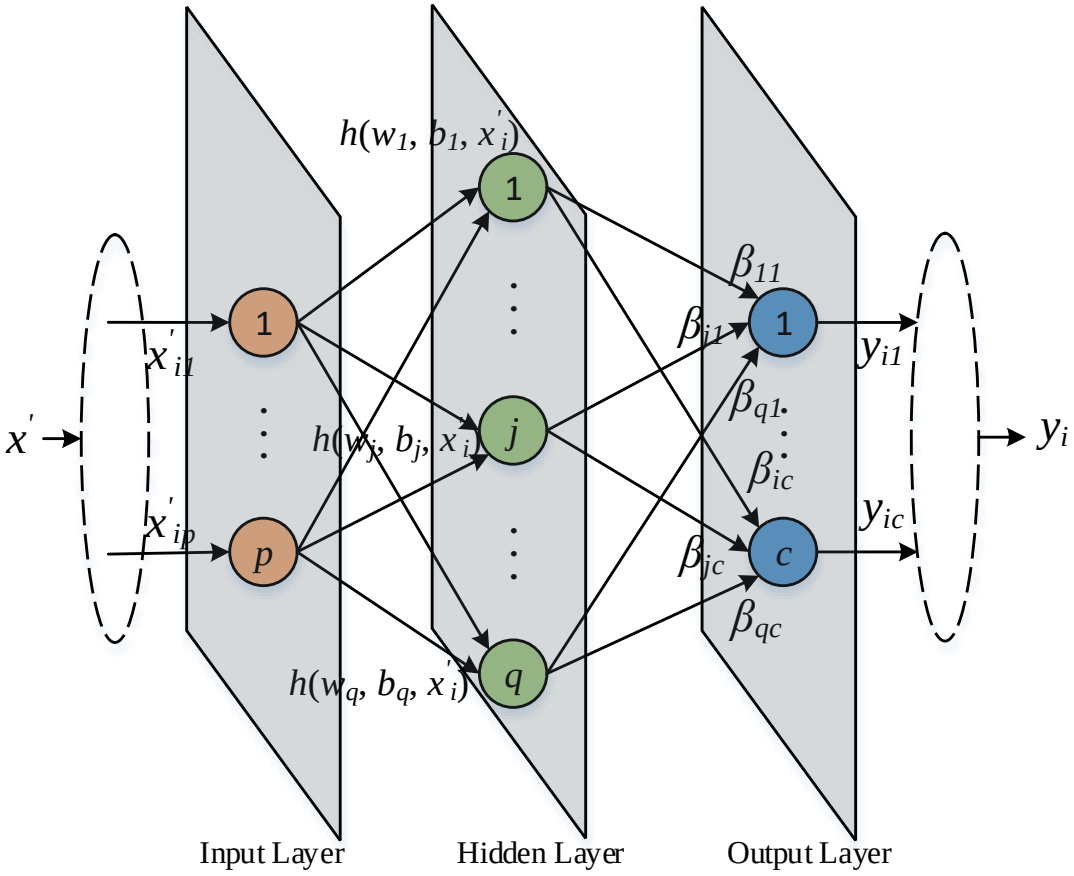


Figure 3.3: The architecture of ELM.

where the i-th row of $\mathbf{H}$ denotes the output vector of the hidden layer with respect to input sample x_i', and the k-th column of $\mathbf{H}$ denotes the output vector of the k-th hidden node with respect to the input samples $x_1', x_2', \ldots, x_n'$.

β denotes the weight matrix connecting the hidden layer and the output layer, which is defined as

$$\beta = \begin{bmatrix} \beta_1^{\mathrm{T}} \\ \vdots \\ \beta_q^{\mathrm{T}} \end{bmatrix}_{q \times c} \tag{3.19}$$

$\mathbf{O}$ denotes the expected label matrix, and each row represents the output vector of one

sample. $\mathbf{O}$ is defined as

$$\mathbf{O} = \begin{bmatrix} {o_1}^{\mathrm{T}} \\ \vdots \\ {o_n}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} o_{11} & \cdots & o_{1c} \\ \vdots & \ddots & \vdots \\ o_{n1} & \cdots & o_{nc} \end{bmatrix}_{n \times c} \tag{3.20}$$

Since the target of training SLFNs is to minimize the output error, i.e., approximating the input samples with zero error as follows

$$\sum_{i=1}^{n} \|o_i - y_i\| = \|\mathbf{O} - \mathbf{Y}\| = 0 \tag{3.21}$$

where $\mathbf{Y} = \begin{bmatrix} {y_1}^{\mathrm{T}} \\ \vdots \\ {y_n}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} y_{11} & \cdots & y_{1c} \\ \vdots & \ddots & \vdots \\ y_{n1} & \cdots & y_{nc} \end{bmatrix}_{n \times c}$ denotes the target output matrix.

Then, we need to solve the following formula

$$\mathbf{H}\beta = \mathbf{Y} \tag{3.22}$$

Huang et al. [50, 49] proved that, for ELM, the weights w_k of the input connection and the bias b_k of the hidden layer node can be randomly and independently designated. Once these parameters are assigned, Eq.(3.22) is converted into a linear system and the output weight matrix β can be analytically determined by finding the least-square solution of the linear system, i.e.,

$$\min_{\beta} \|\mathbf{H}\beta - \mathbf{Y}\| \tag{3.23}$$

The optimal solution of Eq.(3.23) is

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{Y} = (\mathbf{H}^{T}\mathbf{H}) \tag{3.24}$$

where $\mathbf{H}^{\dagger}$ denotes the Moore-Penrose generalized inverse of the hidden layer output matrix $\mathbf{H}$ [112, 59]. The obtained $\hat{\beta}$ can ensure minimum training error, get optimal generalization

ability and avoid plunging into local optimum since $\hat{\beta}$ is unique [50]. This solution can also be obtained with Karush-Kuhn-Tucker (KKT) theorem [30].

Finally, we get the classification function of ELM as

$$f(x') = \mathbf{h}(x')\hat{\beta} = \mathbf{h}(x')\mathbf{H}^{\dagger}\mathbf{Y} \tag{3.25}$$

3.2.3 Model Construction Based on WELM

For imbalanced data, to consider the different importance of the majority class samples (i.e., non-defective modules) and the minority class samples (i.e., defective modules) when building the ELM classifier, we define a $n \times n$ diagonal matrix $\mathbf{W}$ whose diagonal element $\mathbf{W_{ii}}$ denotes the weight of training sample x'_i. More precisely, if x'_i belongs to the majority class, the weight $\mathbf{W_{ii}}$ is relatively lower than the sample that belongs to the minority class. According to the KKT theorem, Eq.(3.24) is rewritten as

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{Y} = (\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{T} \tag{3.26}$$

Then, Eq.(3.25) becomes

$$f(x') = \mathbf{h}(x')\hat{\beta} = \mathbf{h}(x')(\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{T} \tag{3.27}$$

There are mainly two schemes for assigning the weights to the samples of the two classes as follows [180]:

$$\mathbf{W1} = \mathbf{W_{ii}} = \begin{cases} 1/n_P & \text{if } x'_i \in \text{minority class} \\ 1/n_N & \text{if } x'_i \in \text{majority class} \end{cases}, \tag{3.28}$$

or

$$\mathbf{W2} = \mathbf{W_{ii}} = \begin{cases} 0.618/n_P & \text{if } x'_i \in \text{minority class} \\ 1/n_N & \text{if } x'_i \in \text{majority class} \end{cases}, \tag{3.29}$$

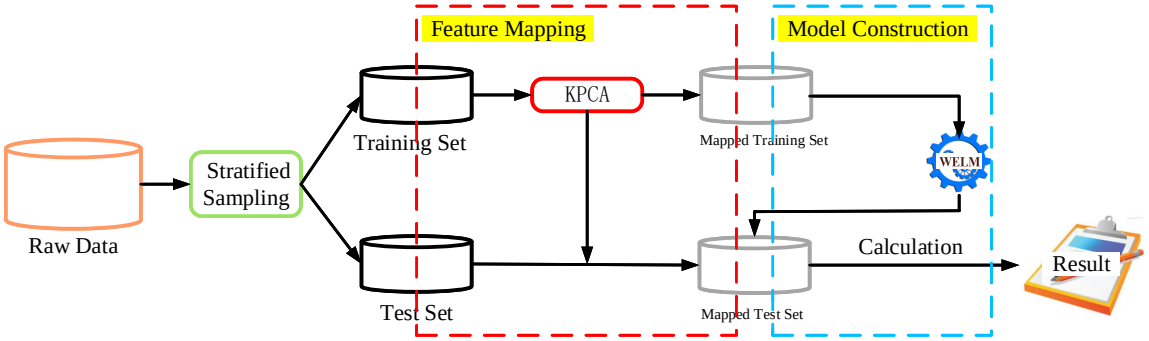


Figure 3.4: Overview of our proposed IVDP framework.

where **W1** and **W2** denote two weighting schemes, n_P and n_N indicate the number of samples of the minority and majority class, respectively. The golden ratio of 0.618:1 between the two classes in scheme **W2** represents the perfection in nature [5].

3.2.4 The Proposed Framework

Figure 3.4 presents our proposed IVDP framework KPWE. The KPCA based feature mapping learning phase is shown in the red rectangle and the WELM based classification model construction and prediction phase is shown in the blue rectangle.

3.3 Study Setup

3.3.1 Research Questions

We design the following four **R**esearch **Q**uestions (**RQ**) to evaluate our KPWE framework.

RQ1: How effective is KPWE compared with basic classifiers with KPCA?

Since our method KPWE combines feature transformation and an advanced classifier, this question is designed to explore the effectiveness of this new classifier compared

against some typical classifiers with the same process of feature extraction.

RQ2: Is KPWE superior to its variants?

Since the two techniques KPCA and WELM used in our framework are variants of the linear feature extraction method PCA and the original ELM respectively, this question is designed to investigate whether our framework is more effective than other combinations of these four techniques.

RQ3: Are the selected features by KPCA more effective for performance improvement than that by other feature selection methods?

To obtain the representative features of the defect data, previous researches [35, 154] used various feature selection methods to select an optimal feature subset to replace the original set. This question is designed to investigate whether the features extracted by KPCA are more effective in improving the defect prediction performance than the features selected by other feature selection methods.

RQ4: Is the prediction performance of KPWE comparable to that of other imbalanced learning methods?

Since our framework KPWE is customized to address the class imbalance issue for software defect data, this question is designed to study whether our framework can achieve better or at least comparable performance than existing imbalanced learning methods.

3.3.2 Benchmark Dataset

To evaluate the performance of our proposed KPWE framework for the IVDP task, we conduct extensive experiments on two public available datasets, including NASA dataset and AEEEM dataset.

NASA dataset is the most popular defect data in previous defect prediction studies [93, 125, 68, 34]. The project data are extracted from a software system or sub-system and consist of a set of static code features, including McCabe complexity, Halstead complexity, and some miscellaneous features. These features are informative predictors to the software quality. The raw NASA dataset is cleaned by Shepperd et al. [125]. Since there are two cleaned versions (D' and D'') of NASA dataset, in this chapter, we use the D'' version as our benchmark dataset as in previous work [34]. For the NASA dataset, a module mainly denotes a function.

This dataset was denoted by D'Ambros et al. [23]. The name comes from the first letter of its five projects, i.e., **A**pache Lucene (LC), **E**quinox (EQ), **E**clipse JDT Core (JDT), **E**clipse PDE UI (PDE), and **M**ylyn (ML). Each project data have 61 features, including 17 source code features, 5 previous-defect features, 5 entropy-of-change features, 17 entropy-of-source-code features, and 17 churn-of-source code features [23]. The linearly decayed entropy based and weighted churn based features are verified to be closely related to defect information. For the AEEEM dataset, a module mainly denotes a class.

Table 3.1 summarize the basic information of the two datasets, including the number of features (# F), the number of modules (# M), the number of defective modules (# DM) and the defect ratios (% DM). Table 3.2 lists the common features of the 10 projects in the NASA dataset. Table 3.3 lists the features contained in each projects in the NASA dataset except for the common features. Table 3.4 lists the features of the 5 projects in the AEEEM dataset.

3.3.3 Evaluation Indicators

In this section, we describe the six evaluation indicators (including three traditional and three effort-aware indicators) used to measure the IVDP performance of our proposed

Table 3.1: Benchmark Datasets for Within Project Defect Prediction

Dataset	Project	# F	# M	# DM	% DM
NASA	CM1	37	327	42	12.84%
	KC1	21	1162	294	25.30%
	KC3	39	194	36	18.56%
	MC1	38	1847	36	1.95%
	MC2	39	125	44	35.20%
	MW1	37	251	25	9.96%
	PC1	37	696	55	7.90%
	PC3	37	1073	132	12.30%
	PC4	37	1276	176	13.79%
	PC5	38	1679	459	27.34%
AEEEM	Equinox	61	324	129	39.81%
	JDT	61	997	206	20.66%
	Lucene	61	691	64	9.26%
	Mylyn	61	1862	245	13.16%
	PDE	61	1497	209	13.96%

framework. In terms of the traditional evaluation indicators, we choose F-measure, MCC, and AUC which are widely used in previous defect prediction studies [149, 102, 58, 155, 111, 71, 93, 40, 57, 133, 146, 153]. In terms of the effort-aware indicators, we employ Effort-Aware Precision (EAP), Effort-Aware Recall (EAR), and Effort-Aware F-measure (EAF) [51, 52].

Table 3.2: The Common Features of the 10 Projects from the NASA Dataset

1.Line count of code	11.Halstead_Volume
2.Count of blank lines	12.Halstead_Level
3.Count of code and comments	13.Halstead_Difficulty
4.Count of comments	14.Halstead_Content
5.Line count of executable code	15.Halstead_Effort
6.number of operators	16.Halstead_Error_Estimate
7.number of operands	17.Halstead_Prog_Time
8.Number of unique operators	18.Cyclomatic_Complexity
9.Number of unique operands	19.Design_Complexity
10.Halstead_Length	20.Essential_Complexity

Table 3.3: The Features Contained in Each Project of the NASA Dataset Except for the Common Ones

Features	CM1	KC1	KC3	MC1	MC2	MW1	PC1	PC3	PC4	PC5
21.Number_of_lines	√		√	√	√	√	√	√	√	√
22.Cyclomatic_Density	√		√	√	√	√	√	√	√	√
23.Branch_Count	√	√	√	√	√	√	√	√	√	√
24.Essential_Density	√		√	√	√	√	√	√	√	√
25.Call_Pairs	√		√	√	√	√	√	√	√	√
26.Condition_Count	√		√	√	√	√	√	√	√	√
27.Decision_Count	√		√	√	√	√	√	√	√	√
28.Decision_Density	√		√		√	√	√	√	√	
29.Design_Density	√		√	√	√	√	√	√	√	√
30.Edge_Count	√		√	√	√	√	√	√	√	√
31.Global_Data_Complexity			√	√	√					√
32.Global_Data_Density			√	√	√					√
33.Maintenance_Severity	√		√	√	√	√	√	√	√	√
34.Modified_Condition_Count	√		√	√	√	√	√	√	√	√
35.Multiple_Condition_Count	√		√	√	√	√	√	√	√	√
36.Node_Count	√		√	√	√	√	√	√	√	√
37.Normalized_CC	√		√	√	√	√	√	√	√	√
38.Parameter_Count	√		√	√	√	√	√	√	√	√
39.Percent_Comments	√		√	√	√	√	√	√	√	√

(1) Traditional Indicators

The three traditional indicators are derived from the basic indicators listed in Table 3.5 and described as follows

F-measure is a trade-off between recall and precision which is defined as

$$\text{F-measure} = \frac{(1+\theta^2) * \text{recall} * \text{precision}}{\theta^2 * \text{precision} + \text{recall}}. \tag{3.30}$$

MCC measures the correlation coefficient between the actual and predicted outputs which is defined as

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})(\text{TN}+\text{FP})(\text{TN}+\text{FN})}}. \tag{3.31}$$

AUC calculates the area under and ROC curve which is a two-dimensional plane with

Table 3.4: The Features of Projects from the AEEEM Dataset

source code features, entropy of source code features, churn of source code features	Coupling between objects (CBO) Depth of inheritance tree (DIT) Number of other classes that reference the class (FanIn) Number of other classes referenced by the class (FanOut) Lack of cohesion in methods (LCOM) Number of children (NOC) Number of attributes (NOA) Number of attributes inherited NOAI Number of lines of code (LOC) Number of methods (NOM) Number of methods inherited (NOMI) Number of private attributes (NOPRA) Number of private methods (NOPRM) Number of public attributes (NOPA) Number of public methods (NOPM) Response for class (NFC) Weighted method count (WMC)
entropy of change features	History of complexity metric (HCM) Exponentially decayed HCM (EDHCM) Linearly decayed HCM (LDHCM) LoGarithmically decayed HCM (LGDHCM) Weighted HCM (WHCM)
previous-defect features	Number of bugs found until Number of critical bugs found untill Number of high priority bugs found until Number of major bugs found until Number of non trivial bugs found until

Table 3.5: Basic Indicators for Defect Prediction

	# Predicted defective	# Predicted non-defective
# Actual defective	**T**rue **P**ositive (**TP**)	**F**alse **N**egative (**FN**)
# Actual non-defective	**F**alse **P**ositive (**FP**)	**T**rue **N**egative (**TN**)
True **P**ositive **R**ate (**TPR**) or recall	$\frac{\mathrm{TP}}{\mathrm{TP+FN}}$	
False **P**ositive **R**ate (**FPR**)	$\frac{\mathrm{FP}}{\mathrm{FP+TN}}$	
precision	$\frac{\mathrm{TP}}{\mathrm{TP+FP}}$	

TPR as the y-axis and FPR as the x-axis.

The θ in Eq.(3.30) is a bias parameter to measure the relative importance of recall and precision. There are three widely-used **F**-measure (**F**) variants, i.e., F_1 ($\theta = 1$) which treats precision and recall equally, $\mathrm{F}_{0.5}$ ($\theta = 0.5$) which prefers precision, and F_2 ($\theta = 2$) which prefers recall. In the defect prediction task, the defective modules are the main concerns and we always want to accurately detect as many defective modules as possible [150]. Thus, the performance measurement should be evaluated bias to this purpose which is consistent with the definition of recall. Thus, in this work, we emphasize more on the importance of recall and choose F_2 variant following the previous studies [55, 56].

Here, we gave an example to illustrate this issue. Assume one defect data with 1000 modules including 200 defective modules and 800 non-defective modules. Two methods are applied to the defect data and obtain the following confusion matrices shown in Table 3.6. Method 1 correctly detects 100 defective modules while Method 2 only correctly detects 50 defective modules. In terms of the two methods, intuitively, the developers are likely to choose Method 1 as it can detect more real defective modules, even though they need to detect more 150 (200-50) modules which are actually non-defective modules if the test resources permit. In this case, for Method 1, $\mathrm{precision} = \frac{100}{100+300} = \frac{1}{4}$, $\mathrm{recall} = \frac{100}{100+100} = \frac{1}{2}$; for Method 2, $\mathrm{precision} = \frac{50}{50+50} = \frac{1}{2}$, $\mathrm{recall} = \frac{50}{50+150} = \frac{1}{4}$. If we used F-measure with $\theta = 1$ to evaluate the performance, we could find that the two methods have the same F-measure value ($\frac{2*1/2*1/4}{1/2+1/4} = \frac{1}{3}$). In this case, we could not determine which method is better. But if we use F-measure with $\theta = 2$ to evaluate the performance, for Method 1, $\mathrm{F\text{-}measure} = \frac{5*1/2*1/4}{4*1/4+1/2} = \frac{5}{12}$; for Method 2, $\mathrm{F\text{-}measure} = \frac{5*1/2*1/4}{4*1/2+1/4} = \frac{5}{18}$. Since $\frac{5}{12} > \frac{5}{18}$, Method 1 is better which is consistent with the initial intuition. From this point of view, choosing F-measure with $\theta = 2$ as the performance indicator is more appropriate than with $\theta = 1$ in the context of defect

prediction scenario. A recent study [3] also pointed out that the F-measure with $\theta = 2$ is a better choice when data is unbalanced.

Table 3.6: Two Confusion Matrices

Method 1				Method 2			
Confusion Matrix		actual label		Confusion Matrix		actual label	
		1	0			1	0
predicted label	1	TP=100	FP=300	predicted label	1	TP=50	FP=50
	0	FN=100	TN=500		0	FN=150	TN=750

(2) Effort-Aware Indicators

The three traditional binary classification indicators do not consider the quality assurance efforts required to review the modules [179, 142, 19]. However, inspecting all the modules is not always practical due to the limited test resources. As suggested by Mende et al. [91], it is more realistic to use effort-aware indicators to evaluate the performance of defect prediction.

Effort-aware indicators evaluate the defect prediction performance within a limited effort required to review the predicted defective modules [4, 61, 166]. In reality, we always want to maximize the profit of any effort for quality assurance. Generally, the efforts denote the LOC that need to be inspected, and the profit is the number or percentage of defective modules discovered. In this thesis, we set the efforts as 20% of total LOC following previous studies [54, 164, 152]. Smaller threshold, such as 5%, 10%, or 15% will be considered in our future work.

To calculate the effort-aware indicators, the general way is to rank the modules according to a specific rule first, then simulate an expert to inspect these modules one at a time in order. In the meanwhile, the percentage of LOC reviewed and the number of detected defective modules are counted. The process is terminated when 20% of total LOC have been inspected. The proportion of detected defective modules among all the

actual defective modules is treated as an effort-aware indicator, which is called **PofB**20 (**P**ercentage **of B**ugs) or **CE**20 (**C**ost **E**ffectiveness) in [54, 152, 168, 165].

In previous studies, researchers ranked the modules in a descending order based on the degree of their predicted risk values. The risk values are defined as the probability outputs of a classification model for the modules [91, 54, 152] or the ratios of the probability outputs to the corresponding LOC [164, 168, 165]. Recently, Huang et al. [51] proposed a novel ranking method to calculate the effort-aware indicators for defective change prediction, called **C**lassifier **B**efore **S**orting (**CBS**). Their experimental results showed that the derived indicator values significantly were improved based on their ranking method. The basic idea of CBS is that among the changes that are predicted to be potentially defective by a classifier, small changes that are measured by the modified LOC should be inspected first, since they give the best bang for the buck [51]. However, this ranking method only ranks the predicted defective changes. It may underestimate the performance since the predicted non-defective changes can also contain actual defective changes. To remedy this limitation, in this thesis, we propose an improved ranking method based on CBS to rank the modules in our IVDP scenario. Figure 3.5 show the calculation process of the effect-aware indicators based on our module ranking method. More specifically, ① we divide the modules into two parts (i.e., the predicted defective and non-defective modules) according to their predicted labels by WELM classifier; ② we rank the predicted defective modules in a ascending order based on their LOC values, this process is identical to CBS. In addition, we also rank the predicted non-defective module with the same process; ③ we concatenate the latter ranking results behind the former ranking results; ④ we obtain the checked modules (i.e., the top nine modules in Figure 3.5) by inspecting 20% of total LOC; ⑤ we count some statistical values to calculate the effort aware indicators.

We concisely describe how to calculate the three effort-aware indicators. Given a current version of a project with n_1 defective modules. After inspecting 20% of total

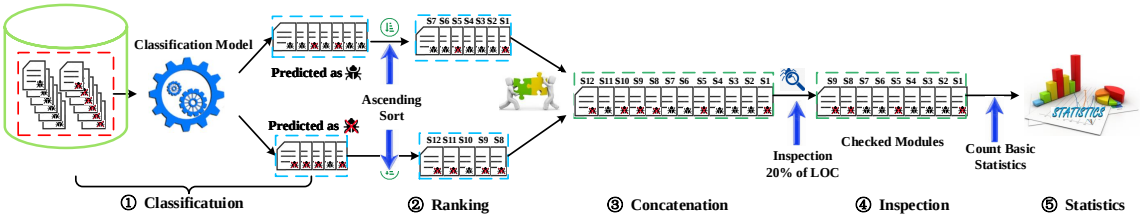


Figure 3.5: The process to calculate the effort aware indicators.

LOC based on our improved ranking strategy, n' modules and ${n_1}'$ defective modules have been inspected.

The first effort-aware indicator is defined as the proportion of the inspected defective modules among all actual defective modules, which is called Recall by Huang et al. [51]. To distinguish it from the traditional Recall indicator, we name it **E**ffort-**A**ware **R**ecall (**EAR**). EAR is defined as

$$\mathrm{EAR} = \frac{{\mathrm{n_1}}'}{\mathrm{n_1}}. \tag{3.32}$$

The second effort-aware indicator is defined as the proportion of the inspected defective modules among all inspected modules, called **E**ffore-**A**ware **P**recision (**EAP**). EAP is defined as

$$\mathrm{EAR} = \frac{{\mathrm{n_1}}'}{\mathrm{n}'}. \tag{3.33}$$

Given the definitions of EAR and EAP, like traditional F-measure indicator, the third effort-aware indicator **E**ffort-**A**ware **F**-measure (**EAF**) is defined as

$$\mathrm{EAF} = \frac{(1+\theta_1^2) * \mathrm{EAR} * \mathrm{EAP}}{\theta_1^2 * \mathrm{EAP} + \mathrm{EAR}}. \tag{3.34}$$

In this thesis, we also set $\theta_1 = 2$ like the setting for the tradition F-measure. The worst case is that EAR (or recall) and EAP (or precision) are all equal to 0. Thus, the value of EAF (or F-measure) makes no sense since the denominator in Eq.(3.34) (or Eq.(3.30)) is

0. In this case, we set the value of EAF (or F-measure) as 0 which indicates the worst IVDP performance.

3.3.4 Parameter Configuration

For the feature mapping, we choose the Guassian RBF as the kernel function of KPCA which achieves promising performance in previous work [70]. The reasons are that this kernel has the advantage to map the samples into a higher dimensional space and can handle the case when the relationship between the class label and features is nonlinear. In addition, it has fewer parameters than other kernels, such as polynomial kernel, which makes the model less complicated. For parameter ω, we set is to 100^2. For model construction, we also choose Guassian RBF as the activation function which is the preferences in previous work [169]. Since there are no theoretical guidances to determine the optimal q for all situations and prior study [50] stated that the q value with far less than the number of training module (i.e., n in this thesis) can usually achieve the best performance, in this work, we set q from 5 to $\frac{n}{2}$ with an increment of 5 to conduct experiments. For the weighting scheme, we choose the second one, i.e., $\mathbf{W2}$.

3.3.5 Inner Version Scenario Setting

For each project, we use the 50:50 split with stratified sampling to constitute the training and test set. More specifically, we utilize stratified sampling to randomly select 50% instances as the training set and the remaining 50% instances as the test set. The stratified sampling strategy guarantees the same defect ratios of the training set and test set which conforms to the actual application scenario. In addition, such sampling setting helps reduce sampling biases [44]. The 50:50 split and stratified sampling are commonly used in previous defect prediction studies [57, 149, 47, 118]. To mitigate the impact of the

random division treatment on the experimental results and produce a general conclusion, we repeat this process 30 times on each project by reshuffling the module order. In this chapter, for the result groups using different q values, we reserve the group corresponding to the best average EAF value. Finally, we report the average values of each indicator across the 30-round experiments.

3.3.6 Statistic Test Method

The reason of using statistic test is that, if the average performance values of multiple methods are different, we still cannot claim that the method with higher average performance is superior to that with lower average performance in the statistical sense as this performance differences may be explained by randomness [42]. In this case, statistic test is used to identify whether the differences in performance between various methods are randomness or statistically significant. In this thesis, we perform the non-parametric Frideman test with the Nemenyi post-hoc test [24] at significant level 0.05 over all cross-version pairs. This test holds no assumption on the distribution of the performance values and is less susceptible to outliers [68, 91, 42]. The Friedman test evaluates whether the average rankings of different methods exist statistically significant differences. The test statistic of the Friedman test can be calculated as follows:

$$\tau_{\chi^2} = \frac{12Q}{L(L+1)}(\sum_{j=1}^{L} AR_j^2 - \frac{L(L+1)^2}{4}), \tag{3.35}$$

where Q denotes the total number of cross-version pairs, L denotes the number of methods needed to be compared, $AR_j = \frac{1}{Q}\sum_{i=1}^{Q} R_i^j$ denotes the average ranking of method j on all cross-version pairs and R_i^j denotes the ranking of jth method on the ith cross-version pair. τ_{χ^2} obeys the χ^2 distribution with $L-1$ degree of freedom [172]. Since the original Friedman test statistic is too conservative, its variant τ_F is usually used to conduct the

statistic test. τ_F is calculated as the following formula:

$$\tau_F = \frac{(Q-1)\tau_{\chi^2}}{Q(L-1)-\tau_{\chi^2}}. \tag{3.36}$$

τ_F obeys the F-distribution with $L-1$ and $(L-1)(Q-1)$ degrees of freedom. Once τ_F value is calculated, we can compare τ_F against critical values for the F distribution and then determine whether to accept or reject the null hypothesis, i.e., all methods perform equally.

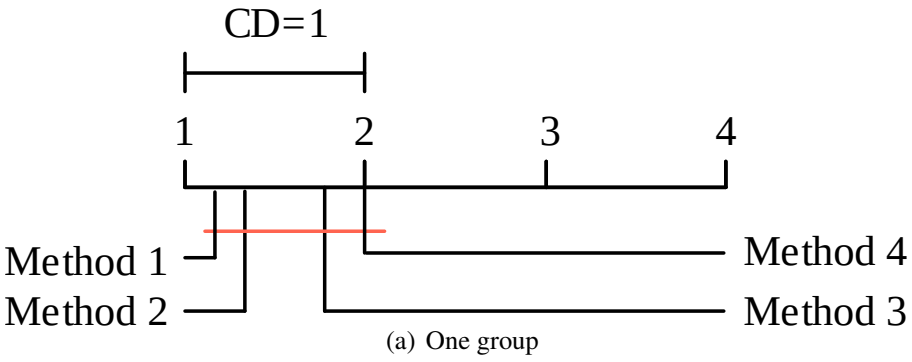
If the null hypothesis is rejected, it means that the performance differences among different methods are nonrandom, then a so-called Nemenyi post-hoc test is performed to check which specific method differs significantly [68]. For each pair of methods, this test uses the average ranking of each method and checks whether the ranking difference exceeds a **C**ritical **D**ifference (**CD**) which is calculated with the following formula:

$$CD = q_{\alpha,L}\sqrt{\frac{L(L+1)}{6Q}}, \tag{3.37}$$

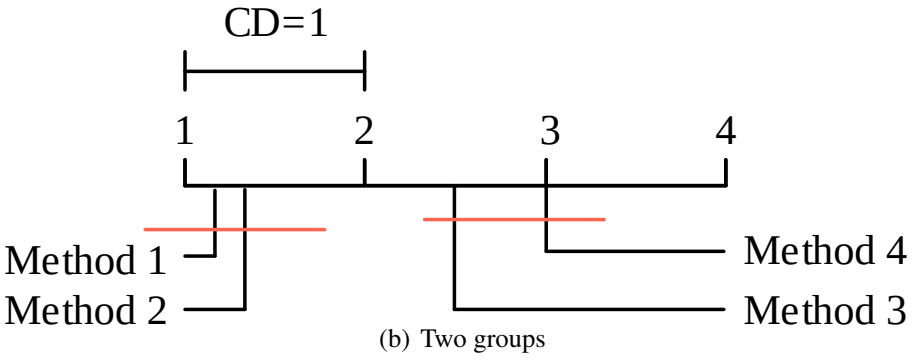
where $q_{\alpha,L}$ is a critical value that related to the method number L and the significance level α. The Frideman test with the Nemenyi post-hoc test is widely used in previous studies [68, 91, 55, 23, 101, 77, 74].

However, the main drawback for post-hoc Nemenyi test is that it may generate overlapping groups for the methods that are compared, which means that a method may belong to multiple significantly different groups [34, 42]. In this thesis, we utilize the strategy in [42] to address this issue. Figure 3.6 depicts an example of the division rules as follows.

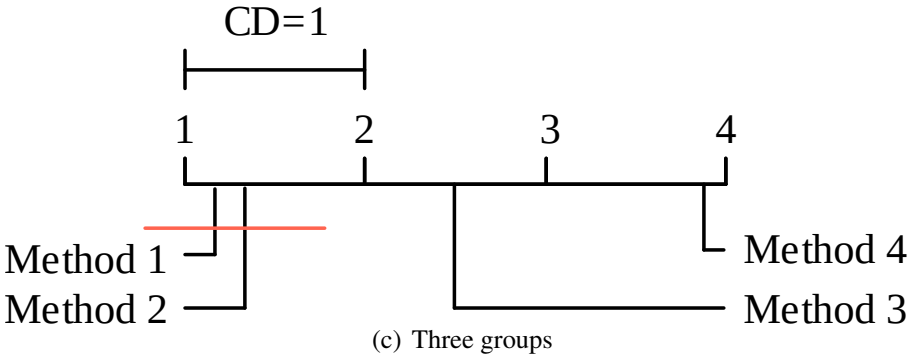
The post-hoc Nemenyi test divides the methods to a same group without significant differences if the discrepancy of their ranks is lower that CD value. Thus, in Figure 3.6(a),



(a) One group



(b) Two groups



(c) Three groups

Figure 3.6: An example of group division for Nemenyi test.

as the discrepancy of the best rank (1.2 for Method 1) and the worse rank (2 for Method 4) among the method set is less than CD value, all methods belong to one group (linked with red line).

In Figure 3.6(b), as the discrepancy of the best rank (1.2 for Method 1) and the worse rank (3 for Method 4) among the method set is more than CD value but less than 2 CD values, these methods belong to 2 groups. More specifically, the method (Method 2)

belongs to the top group because its rank (0.4) is closer to the best rank. Similarly, Method 3 belongs to the bottom group because its rank (2.5) is closer to the worst rank.

In Figure 3.6(c), as the discrepancy of the best rank (1.2 for Method 1) and the worse rank (3.8 for Method 4) among the method set is more than 2 CD values, these methods belong to 3 groups. More specifically, Method 2 belongs to the top group because its rank discrepancy to the best rank is less than CD value. Only Method 4 belongs to the bottom group because there exist no methods whose rank discrepancy to the worst rank is less than CD value. Other methods (i.e., Method 3) whose rank discrepancies to the best rank and worst rank are all larger than CD value belong to the middle group.

Using the rules, the generated groups are non-overlapping with significantly differences.

3.4 Experimental Results

3.4.1 Results for RQ1

Methods: Since many previous defect prediction studies applied classic classifiers as prediction models [68, 34], in this work, we choose seven representative classifiers, including **N**aive **B**ayes (**NB**), **N**earest **N**eighbor (**NN**), **R**andom **F**orest (**RF**), **L**ogistic **R**egression (**LR**), **C**lassification **a**nd **R**egression **T**ree (**CART**).

Results: Table 3.7 presents the average indicator values of KPCA with six classifiers on NASA dataset, AEEEM dataset, and across the two datasets (TOTAL). Note that KPNB means the method combines KPCA and NB classifier. Figure 3.7 depicts the statistic test results on the 15 projects across the two datasets. From Table 3.7 and Figure 3.7, we have the following observations.

First, from Table 3.7, on the 10 projects of NASA dataset, our proposed KPWE

Table 3.7: Average Indicator Values of KPCA with Six Classifiers on Each Dataset and Across All Datasets

Indicator	Dataset	KPNB	KPNN	KPLR	KPCARF	KPRF	KPWE
F	NASA	0.262	0.306	0.245	0.298	0.190	**0.448**
	AEEEM	0.453	0.403	0.365	0.390	0.283	**0.476**
	TOTAL	0.326	0.338	0.285	0.329	0.221	**0.457**
MCC	NASA	0.212	0.174	0.232	0.164	0.167	**0.272**
	AEEEM	0.333	0.273	**0.344**	0.250	0.271	0.329
	TOTAL	0.252	0.207	0.269	0.193	0.202	**0.291**
AUC	NASA	0.714	0.586	**0.742**	0.611	0.689	0.721
	AEEEM	0.762	0.635	**0.772**	0.651	0.753	0.715
	TOTAL	0.730	0.602	**0.752**	0.625	0.710	0.719
EAP	NASA	0.313	0.238	0.284	0.238	0.206	**0.336**
	AEEEM	**0.419**	0.336	0.366	0.325	0.298	0.417
	TOTAL	0.348	0.270	0.311	0.267	0.237	**0.363**
EAR	NASA	0.232	0.306	0.259	0.291	0.258	**0.419**
	AEEEM	0.349	0.358	0.348	0.342	0.359	**0.408**
	TOTAL	0.271	0.323	0.288	0.308	0.291	**0.415**
EAF	NASA	0.219	0.257	0.214	0.250	0.193	**0.377**
	AEEEM	0.353	0.325	0.297	0.316	0.279	**0.402**
	TOTAL	0.264	0.280	0.242	0.272	0.222	**0.386**

framework achieves the best performance in terms of five indicators (except for AUC); on the five projects of AEEEM dataset, our proposed KPWE framework achieves the best performance in terms of three indicators (except for MCC, AUC, and EAP); across all above 15 projects, our proposed KPWE framework achieves the best performance in terms of five indicators (except for AUC).

Second, compared with the five baseline methods, for the average indicator values of KPWE across the two datasets, KPWE achieves average improvements of 56.3%, 31.9%, 21.0%, 40.5%, and 52.0% in terms of F, MCC, EAP, EAR, and EAF, respectively. Compared with the best average indicator values among the 5 baseline methods, KPWE achieves average improvements of 39.1%, 8.2%, 4.1%, 28.3%, and 38.0% in terms of F,

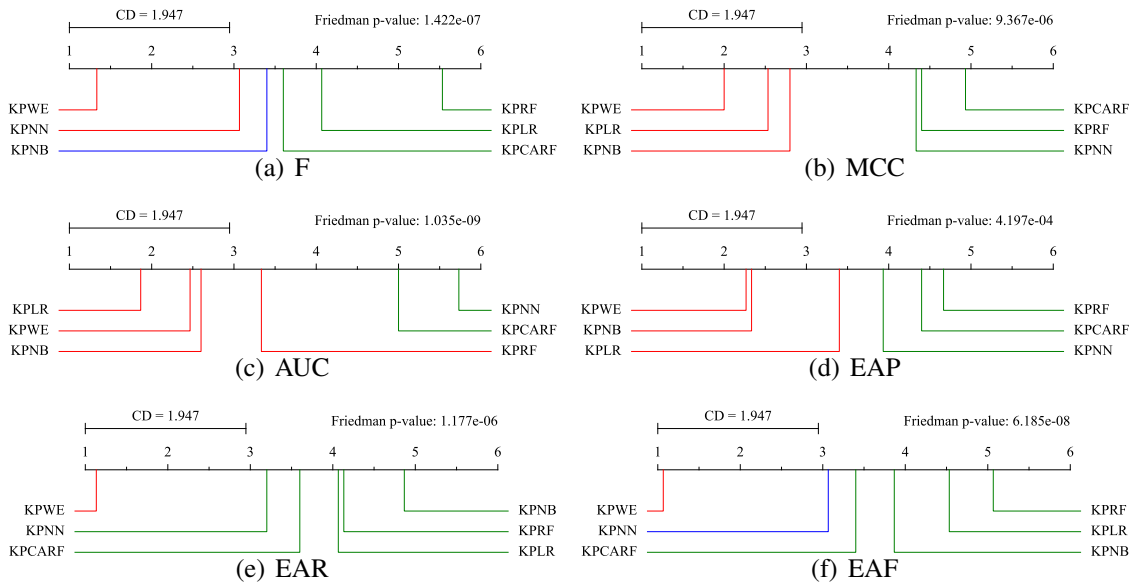


Figure 3.7: Comparison of KPWE and other five basic classifiers with Friedman test and Nemenyi post-hoc test in terms of all six indicators.

MCC, EAP, EAR, and EAF, respectively. In addition, the baseline method KPLR achieves the best average AUC value which is 4.6% better than our KPWE framework.

Third, from Figure 3.7, we observe that the p values of Friedman test in all subfigures are all less than 0.05, which means that there exist significant performance differences among the six methods. Our proposed KPWE framework always belongs to the top-ranked group in terms of all indicators and ranks the first in terms of five indicators (except for AUC). In addition, KPWE is significantly superior to all five baseline methods in terms of two indicators, i.e., EAR and EAF. But KPWE has no significant differences compared with 1, 2, 3, and 2 baseline methods in terms of F, MCC, AUC, and EAP, respectively.

Discussion: The reason why KPWE performs better than the 5 baseline methods is that it uses an advanced classifier to consider the class imbalance issue of the software defect data while traditional classifiers could not well copy with the imbalanced data.

Answer to RQ1: To sum up, the classifier that considers the class imbalance issue can

work better on the defect data mapped by KPCA. It means that the class imbalance of the defect data in the data can hinder the model to obtain better defect prediction performance.

3.4.2 Results for RQ2

Methods: To answer this question, we first compare KPWE against the baseline methods that combine WELM with PCA (short for PCAWELM) and none feature extraction (short for WELM). It can be used to investigate the performance differences among the methods using non-linear, linear and none feature extraction for WELM. Then, we compare KPWE against the baseline methods that combine ELM with KPCA, PCA, and none feature extraction (short for KPCAELM, PCAELM, and ELM respectively). It can be used to compare the performance differences of our framework against its downgraded version methods that do not consider the class imbalance issue. All these baseline methods are treated as the variants of KPWE.

Results: Table 3.8 presents the average indicator values of KPWE and its 5 variant methods on NASA dataset, AEEEM dataset, and across the two datasets. Table 3.9 reports the node number of the hidden layer for the corresponding WELM and ELM classifiers of the six methods when obtaining the optimal EAF values. Figure 3.8 depicts the statistic test results on the 15 projects across the two datasets. From Table 3.8, Table 3.9, and Figure 3.8, we have the following observations.

First, from Table 3.8, on the 10 projects of NASA dataset, our proposed KPWE framework achieves the best performance in terms of all indicators; on the five projects of AEEEM dataset, our proposed KPWE framework achieves the best performance in terms of all indicators; across all above 15 projects, our proposed KPWE framework also achieves the best performance in terms of all indicators.

Table 3.8: Average Indicator Values of KPWE and Its Five Variant Methods on Each Dataset and Across All Datasets

Indicator	Dataset	ELM	PCAELM	KPCAELM	WELM	PCAWELM	KPWE
F	NASA	0.253	0.241	0.327	0.357	0.366	**0.448**
	AEEEM	0.255	0.274	0.349	0.380	0.394	**0.476**
	TOTAL	0.254	0.252	0.334	0.365	0.375	**0.457**
MCC	NASA	0.119	0.119	0.206	0.143	0.156	**0.272**
	AEEEM	0.167	0.188	0.252	0.177	0.201	**0.329**
	TOTAL	0.135	0.142	0.221	0.154	0.171	**0.291**
AUC	NASA	0.622	0.619	0.653	0.632	0.642	**0.721**
	AEEEM	0.671	0.678	0.629	0.643	0.660	**0.715**
	TOTAL	0.639	0.638	0.645	0.636	0.648	**0.719**
EAP	NASA	0.186	0.186	0.304	0.229	0.235	**0.336**
	AEEEM	0.234	0.247	0.368	0.279	0.294	**0.417**
	TOTAL	0.202	0.207	0.325	0.246	0.254	**0.363**
EAR	NASA	0.297	0.304	0.308	0.333	0.337	**0.419**
	AEEEM	0.369	0.369	0.343	0.345	0.354	**0.408**
	TOTAL	0.321	0.326	0.320	0.337	0.343	**0.415**
EAF	NASA	0.235	0.238	0.278	0.299	0.291	**0.377**
	AEEEM	0.277	0.282	0.313	0.338	0.327	**0.402**
	TOTAL	0.249	0.253	0.290	0.312	0.303	**0.386**

Second, compared with the five baseline methods, for the average indicator values of KPWE across the two datasets, KPWE achieves average improvements of 49.1%, 82.1%, 21.0%, 51.4%, 26.1%, and 38.4% in terms of F, MCC, AUC, EAP, EAR, and EAF, respectively. Compared with the best average indicator values among the 5 baseline methods, KPWE achieves average improvements of 21.9%, 31.6%, 11.0%, 11.6%, 21.1%, and 23.7% in terms of F, MCC, AUC, EAP, EAR, and EAF, respectively.

Third, from Table 3.9, we observe that our proposed KPWE needs the fewest node number of the hidden layer on 8 out of 10 project of NASA dataset and 4 out of 5 projects of AEEEM dataset. This means that the mapped data preprocessed by KPCA can achieves better prediction performance with fewer node number of hidden layer. In addition, using fewer nodes of the hidden layer means that there are fewer parameters of the network that

Table 3.9: The Needed Node Number of Hidden Layer for KPWE and It Five Variants

Dataset	Project	ELM	PCAELM	KPCAELM	WELM	PCAWELM	KPWE
NASA	CM1	110	140	110	80	55	**10**
	KC1	150	145	110	110	150	**80**
	KC3	115	**10**	40	45	35	40
	MC1	90	115	150	90	85	**30**
	MC2	125	65	80	135	130	**55**
	MW1	60	70	50	30	20	**10**
	PC1	150	145	100	130	140	**35**
	PC3	145	150	130	100	70	**40**
	PC4	150	140	140	135	110	**95**
	PC5	**5**	**5**	150	105	145	145
AEEEM	EQ	90	75	120	85	**65**	95
	JDT	140	140	150	**135**	140	**135**
	LC	75	115	130	95	95	**10**
	ML	145	105	140	150	150	**85**
	PDE	150	150	150	110	145	**25**

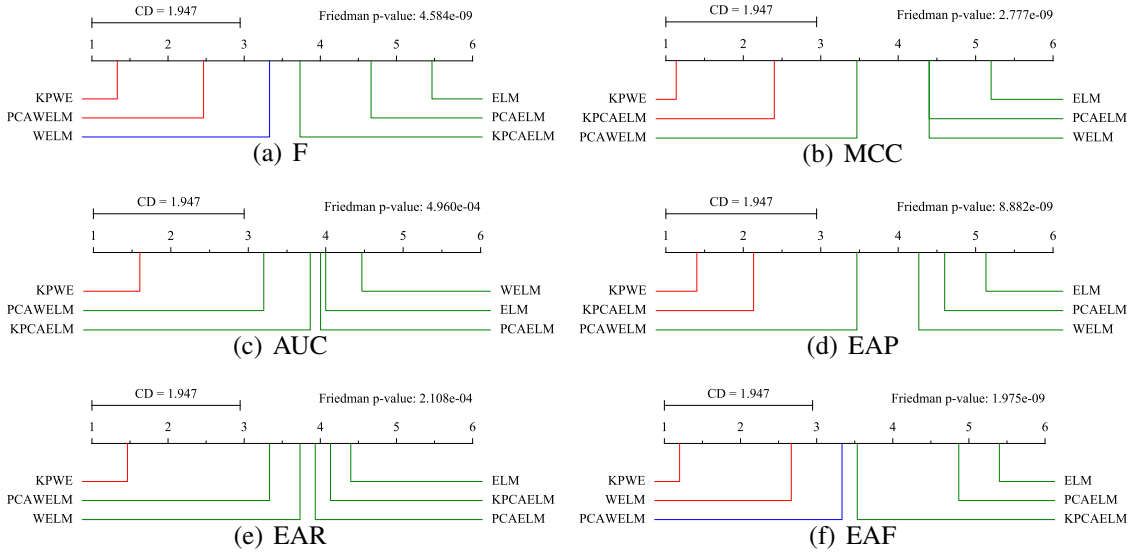


Figure 3.8: Comparison of KPWE and its five variant methods with Friedman test and Nemenyi post-hoc test in terms of all six indicators.

are needed to learn. This also implies that the corresponding model is more lightweight.

Fourth, from Figure 3.8, we observe that the p values of Friedman test in all subfigures are all less than 0.05, which means that there exist significant performance differences among the six methods. Our proposed KPWE framework always belongs to the top-ranked group and ranks the first in terms of all indicators. In addition, KPWE is significantly superior to all five baseline methods in terms of two indicators, i.e., AUC and EAR. But KPWE has no significant differences compared with only one baseline method in terms of F, MCC, EAP, and EAF, respectively.

Discussion: On the one hand, from the average performance across the two datasets, KPWE and KPCAELM are superior to PCAWELM and PCAELM in terms of all 6 indicators respectively. On the other hand, KPWE and KPCAELM perform better than WELM and ELM respectively. All these mean that the features extracted by the nonlinear method KPCA are beneficial to ELM and WELM for the improvement of defect prediction performance compared against the raw features or the features extracted by linear method PCA. Moreover, KPWE, PCAWELM and WELM are superior to KPCAELM, PCAELM and ELM respectively, which denotes that WELM is more appropriate to the class imbalanced defect data than ELM for performance improvement.

Answer to RQ2: In sum, the mapped features by the nonlinear extraction method is superior to the original features and features by the linear extraction method. In addition, the weighted ELM (i.e., WELM) is more suitable to work well on imbalanced defect data than the original ELM.

3.4.3 Results for RQ3

Methods: To answer this question, we choose eight representative feature selection methods, include four filter-based feature ranking methods and four wrapper-based feature subset selection methods, for comparison. The filter-based methods are **C**hi-**S**quare (**CS**), **F**ish **S**core (**FS**), **I**nformation **G**ain (**IG**) and **Re**lief**F** (**ReF**). The first two methods are both based on statistics, and the last two are based on entropy and instance, respectively. These methods have been proven to be effective for defect prediction [154, 126]. For wrapper-based methods, we choose four commonly-used classifiers (i.e., NB, NN, LR, and RF) and F-measure to evaluate the performance of the selected feature subset. The four wrapper methods are abbreviated as NBWrap, NNWrap, LRWrap, and RFWrap, respectively. Following the previous work [154, 33], we set the number of selected features to $\lceil log_2 m \rceil$, where m is the number of original features.

Table 3.10: Average Indicator Values of KPWE and Other Eight Feature Selection Methods with WELM on Each Dataset and Across All Datasets

Indicator	Dataset	CS	FS	IG	ReF	NBWrap	NNWrap	LRWrap	RFWrap	KPWE
F	NASA	0.337	0.392	0.337	0.404	0.390	0.418	0.406	0.407	**0.448**
	AEEEM	0.291	0.468	0.279	0.438	0.449	0.449	0.436	0.464	**0.476**
	TOTAL	0.322	0.417	0.318	0.416	0.409	0.429	0.416	0.426	**0.457**
MCC	NASA	0.144	0.192	0.129	0.222	0.198	0.223	0.214	0.223	**0.272**
	AEEEM	0.049	0.322	0.048	0.288	0.282	0.300	0.294	0.306	**0.329**
	TOTAL	0.112	0.235	0.102	0.244	0.226	0.249	0.241	0.251	**0.291**
AUC	NASA	0.619	0.651	0.600	0.679	0.661	0.684	0.671	0.676	**0.721**
	AEEEM	0.532	**0.752**	0.532	0.702	0.705	0.717	0.710	0.730	0.715
	TOTAL	0.590	0.685	0.577	0.686	0.675	0.695	0.684	0.694	**0.719**
EAP	NASA	0.232	0.271	0.220	0.291	0.264	0.287	0.283	0.287	**0.336**
	AEEEM	0.186	0.382	0.190	0.364	0.356	0.367	0.373	0.372	**0.417**
	TOTAL	0.216	0.308	0.210	0.315	0.295	0.313	0.313	0.315	**0.363**
EAR	NASA	0.308	0.364	0.310	0.370	0.357	0.374	0.360	0.372	**0.419**
	AEEEM	0.362	0.359	0.367	0.364	0.364	0.378	0.360	0.371	**0.408**
	TOTAL	0.326	0.362	0.329	0.368	0.359	0.376	0.360	0.372	**0.415**
EAF	NASA	0.268	0.323	0.266	0.333	0.314	0.335	0.326	0.331	**0.377**
	AEEEM	0.265	0.352	0.263	0.352	0.351	0.357	0.348	0.361	**0.402**
	TOTAL	0.267	0.333	0.265	0.339	0.326	0.342	0.334	0.341	**0.386**

Table 3.11: The Needed Node Number of Hidden Layer for KPWE and Eight Feature Selection Methods with WELM

Dataset	Project	CS	FS	IG	ReF	NBWrap	NNWrap	LRWrap	RFWrap	KPWE
NASA	CM1	30	80	60	65	45	40	55	55	**10**
	KC1	135	150	145	**70**	130	105	150	150	80
	KC3	45	25	50	20	40	30	**15**	**15**	40
	MC1	**10**	15	20	25	40	30	20	50	30
	MC2	**30**	120	100	80	35	135	60	40	55
	MW1	25	25	20	**10**	**10**	20	**10**	30	**10**
	PC1	90	65	100	75	100	75	70	**35**	**35**
	PC3	45	65	95	35	30	55	**15**	35	40
	PC4	**55**	115	60	90	65	65	65	85	95
	PC5	150	135	**105**	125	130	125	135	130	145
AEEEM	EQ	110	**15**	55	50	65	35	30	35	95
	JDT	55	**15**	20	60	35	55	25	35	135
	LC	30	**10**	40	15	15	20	30	15	**10**
	ML	100	**15**	100	140	90	50	110	55	85
	PDE	60	40	30	40	50	**20**	30	25	25

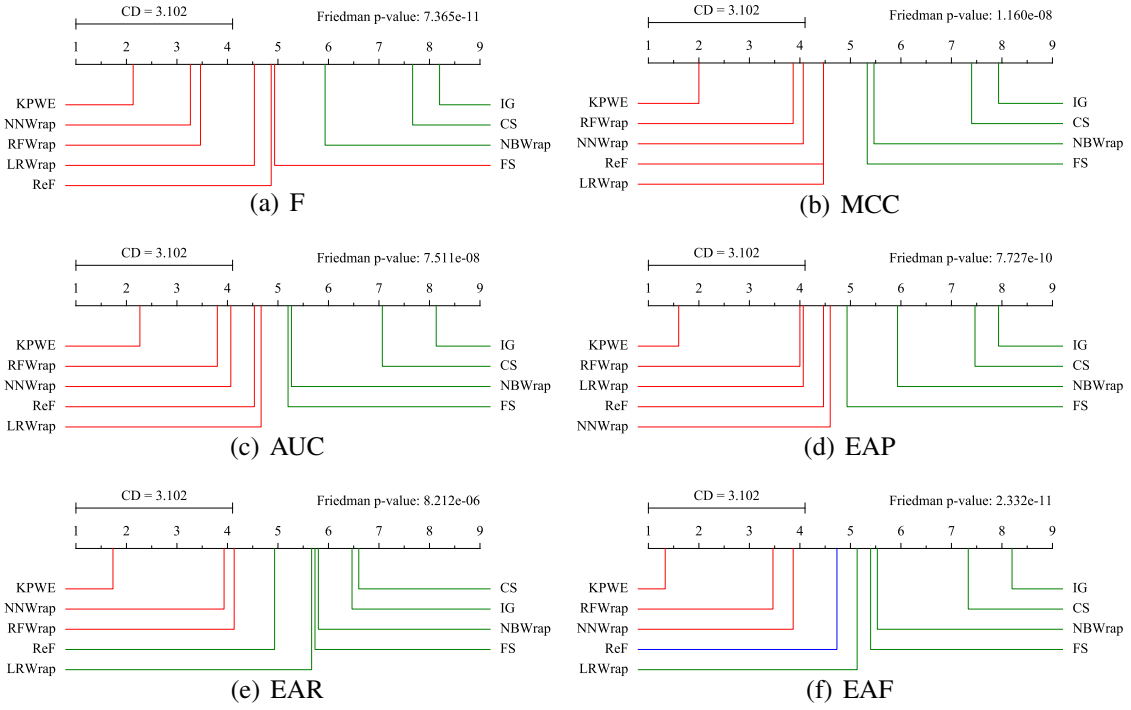


Figure 3.9: Comparison of KPWE and other eight feature selection methods with WELM with Friedman test and Nemenyi post-hoc test in terms of all six indicators.

Results: Table 3.10 presents the average indicator values of KPWE and WELM with eight feature selection methods on NASA dataset, AEEEM dataset, and across the two datasets. Table 3.11 reports the node number of the hidden layer for the corresponding WELM classifiers of the nine methods when obtaining the optimal EAF values. Figure 3.9 depicts the statistic test results on the 15 projects across the two datasets. From Table 3.10, Table 3.11, and Figure 3.9, we have the following observations.

First, from Table 3.10, on the 10 projects of NASA dataset, our proposed KPWE framework achieves the best performance in terms of all indicators; on the five projects of AEEEM dataset, our proposed KPWE framework achieves the best performance in terms of five indicators (except for AUC); across all above 15 projects, our proposed KPWE framework achieves the best performance in terms of all indicators.

Second, compared with the eight baseline methods, for the average indicator values of KPWE across the two datasets, KPWE achieves average improvements of 17.6%, 58.7%, 9.3%, 30.4%, 16.7%, and 22.5% in terms of F, MCC, AUC, EAP, EAR, and EAF, respectively. Compared with the best average indicator values among the 8 baseline methods, KPWE achieves average improvements of 7.3%, 15.9%, 3.5%, 15.1%, 10.5%, and 12.8% in terms of F, MCC, AUC, EAP, EAR, and EAF, respectively.

Third, Table 3.11 shows that, for the 10 projects in the NASA dataset, there is no specific method that can achieve the best performance values on most projects with the lowest node number of hidden layer. For the five projects in the AEEEM dataset, the FS method obtains the best performance values with the least node number of hidden layer on four projects, but this method does not obtain the best performance value with the least number of hidden layer nodes on any project from the NASA dataset. This implies that on the defect data preprocessed by these feature selection methods, the method that can obtain the best performance value with the least hidden layer node varies on different projects.

Fourth, from Figure 3.9, we observe that the p values of Friedman test in all subfigures are all less than 0.05, which means that there exist significant performance differences among the nine methods. Our proposed KPWE framework always belongs to the top-ranked group and ranks the first in terms of all indicators. In addition, KPWE has no significant differences compared with 5, 4, 4, 4, 2, and 2 baseline methods in terms of F, MCC, AUC, EAP, EAR, and EAF, respectively.

Discussion: The reason why the features extracted by KPCA are more effective is that, the eight feature selection methods only select a subset of original features that are not able to excavate the important information hidden behind the raw data, whereas KPCA can eliminate the noise in the data and extract the intrinsic structures of the data that are more helpful to distinguish the class labels of the modules.

Answer to RQ3: To sum up, compared with the feature subset selected from the original features, the mapped features by the nonlinear extraction method can better reveal the important structural information of the defect data, which makes it easier to distinguish the software modules with different labels.

3.4.4 Results for RQ4

Methods: To answer this question, we employ three classic imbalanced learning methods based on data sampling strategies, including **R**andom **U**nder-**S**ampling (**RUS**), **R**andom **O**ver-**S**ampling (**ROS**) or SMOTE techniques to rebalance the modules of the two classes in the training set. Also, we employ two widely-used ensemble learning methods (i.e., **Bag**ging (**Bag**) and **Ada**boost (**Ada**)) for comparison. Moreover, we use another imbalanced learning method **A**symmetric **P**artial **L**east **S**quares **C**lassifier (**APLSC**) [110] as one of the baseline methods.

Table 3.12: Average Indicator Values of KPWE and Other Six Imbalanced Learning Methods on Each Dataset and Across All Datasets

Indicator	Dataset	ROS	RUS	SMOTE	Bagging	AdaBoost	APLSC	KPWE
F	NASA	0.382	0.455	0.445	0.232	0.265	**0.510**	0.448
	AEEEM	0.448	0.529	0.503	0.388	0.383	**0.567**	0.476
	TOTAL	0.404	0.480	0.465	0.284	0.305	**0.529**	0.457
MCC	NASA	0.225	0.186	0.218	0.224	0.214	0.270	**0.272**
	AEEEM	0.294	0.263	0.280	**0.385**	0.331	0.346	0.329
	TOTAL	0.248	0.211	0.239	0.278	0.253	**0.296**	0.291
AUC	NASA	0.626	0.636	0.646	0.753	0.736	**0.758**	0.721
	AEEEM	0.656	0.671	0.670	**0.806**	0.773	0.783	0.715
	TOTAL	0.636	0.648	0.654	**0.771**	0.748	0.767	0.719
EAP	NASA	0.288	0.183	0.233	0.242	0.250	0.229	**0.336**
	AEEEM	0.354	0.250	0.297	0.371	0.366	0.320	**0.417**
	TOTAL	0.310	0.205	0.254	0.285	0.289	0.259	**0.363**
EAR	NASA	0.365	0.350	0.374	0.260	0.274	0.367	**0.419**
	AEEEM	0.382	**0.429**	0.413	0.371	0.357	0.425	0.408
	TOTAL	0.370	0.376	0.387	0.297	0.301	0.386	**0.415**
EAF	NASA	0.317	0.274	0.312	0.202	0.223	0.306	**0.377**
	AEEEM	0.361	0.350	0.369	0.315	0.310	0.383	**0.402**
	TOTAL	0.332	0.299	0.331	0.240	0.252	0.331	**0.386**

Results: Table 3.12 presents the average indicator values of KPWE and six imbalanced learning methods on NASA dataset, AEEEM dataset, and across the two datasets. Figure 3.10 depicts the statistic test results on the 15 projects across the two datasets. From Table 3.12 and Figure 3.10, we have the following observations.

First, from Table 3.12, on the 10 projects of NASA dataset, our proposed KPWE framework achieves the best performance in terms of three effort-aware indicators; on the five projects of AEEEM dataset, our proposed KPWE framework achieves the best performance in terms of two effort-aware indicators (except for EAR); across all above 15 projects, our proposed KPWE framework achieves the best performance in terms of three effort-aware indicators.

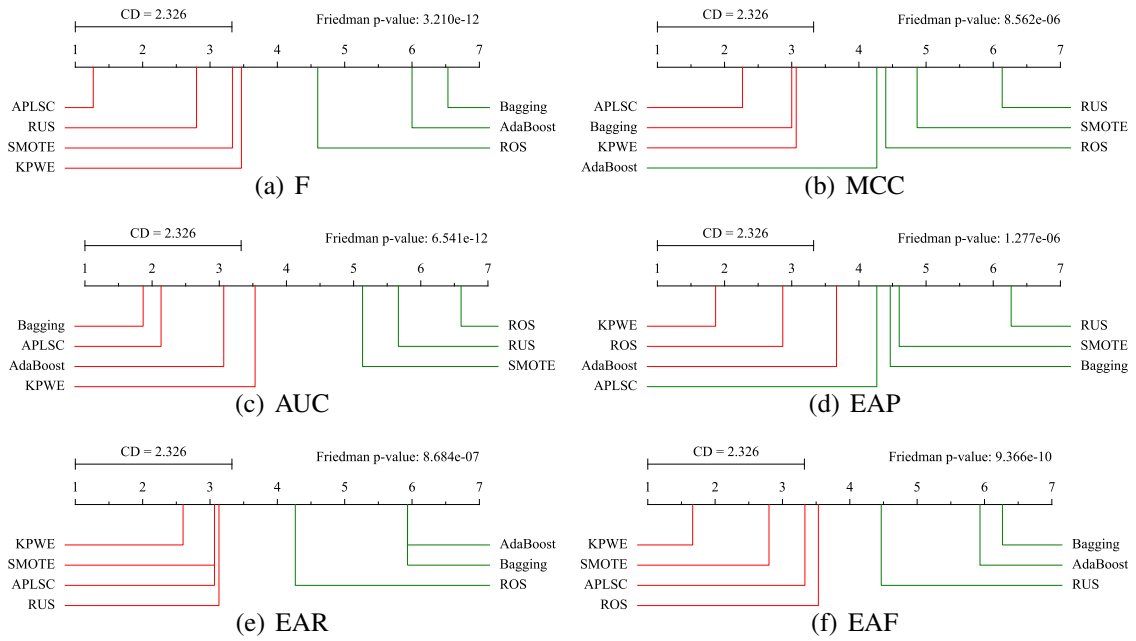


Figure 3.10: Comparison of KPWE and other six imbalanced learning methods with Friedman test and Nemenyi post-hoc test in terms of all six indicators.

Second, compared with the six baseline methods, for the average indicator values of KPWE across the two datasets, KPWE achieves average improvements of 38.3%, 19.1%, and 32.1% in terms of EAP, EAR, and EAF, respectively. Compared with the best average indicator values among the six baseline methods, KPWE achieves average improvements of 17.1%, 7.3%, and 16.4% in terms of EAP, EAR, and EAF, respectively. In addition, APLSC method achieves the best average F and MCC values, Bagging method achieves the best average AUC value.

Third, from Figure 3.10, we observe that the p values of Friedman test in all subfigures are all less than 0.05, which means that there exist significant performance differences among the nine methods. Our proposed KPWE framework always belongs to the top-ranked group in terms of all indicators and ranks the first in terms of three effort-aware indicators. In addition, KPWE has no significant differences compared with 3, 2, 3, 2, 3, and 3 baseline methods in terms of F, MCC, AUC, EAP, EAR, and EAF, respectively.

Discussion: The under-sampling methods may neglect the potentially useful information contained in the ignored non-defective modules, and the over-sampling methods may cause the model over-fitting by adding some redundancy defective modules. Besides, data sampling based imbalanced learning methods usually change the data distribution of the defect data. From this point, the learning methods (such as our KPWE framework) which does not change the data distribution are better choices for imbalanced defect data. In addition, the ensemble based imbalanced learning methods are sensitive to the noises in the data, while our proposed KPWE framework can eliminate the data noises to a certain extent by using the KPCA based feature extraction method.

Answer to RQ4: In sum, KPWE is superior to almost all 6 baseline methods (except for EAR indication on AEEEM dataset), while Bagging and APLSC perform better in terms of the traditional indicators.

3.5 Conclusion

In this chapter, we propose an IVDP framework KPWE combining a feature representation learning method and imbalanced learning method. First, KPWE employs the KPCA method to map the original features into a latent space in which the modules that are linearly indivisible become easily divisible. Then, KPWE uses the WELM method to construct the classification model on the mapped data to predict the labels of the test data. We compare our KPWE framework with five classic classifiers, five variants, eight feature selection methods and six imbalanced learning methods on 10 projects of NASA dataset and five projects of AEEEM dataset. We use three tradition and three effort-aware indicators to evaluate the performance of these methods. The experimental results show that our proposed KPWE framework can achieve better performance in most cases.

Chapter 4

A Two-Stage Training Subset Selection Framework for Cross Version Defect Prediction

Many existing defect prediction work mainly studied the performance of different machine learning methods for the IVDP task [68, 34, 35]. For the mature projects with multiple historical versions, it is a more realistic application scenarios to use the labeled data of previous versions to predict the module labels of the version under development, i.e., CVDP [83, 162]. However, there are not many studies on this topic.

4.1 Motivation

CVDP has some unique characteristics over IVDP. For example, in IVDP scenario, as the training set and test set are usually derived from the same defect data of a specific project version, the data distributions of the two sets are identical, which conforms to the similar distribution assumption of the training set and test set for most classification models [10]. By contrast, in CVDP scenario, as the software functions are increasingly complicated, the modules undergo frequent changes during the version update. For example, the current

version of the project inherits, refactories and deletes some existing modules from the prior version, or adds some new modules [18, 83]. These operations can cause a certain degree of distribution differences of the data across versions. Such distribution differences may result in the CVDP model built on the data of the prior version failing to achieve satisfactory prediction performance for the current version.

To narrow the gap of the distribution differences between the cross-version data, we propose a novel **T**wo-**S**tage **T**raining **S**ubset **S**election (**TSTSS**) method to select a module subset from the prior version that is optimal for the data of the current version. More specifically, in the first stage, TSTSS employs a **S**parse **M**odeling **R**epresentative **S**election (**SMRS**) method [144] to simplify the data of the prior version by selecting an important module subset. The goal is to find a compact representation of the data in the prior version and expect that these important modules can reconstruct the original data to the maximum extent. In addition, the advantage of this simplification is beneficial to save memory, improve the understanding and interpretation of the data [144]. As this step retains the important modules towards the whole data and eliminates the unprofitable modules, it is also helpful to improve the performance of models built on this module subset and save the computational time. Since this module reduction process just selects a module subset to well construct the original data without involving the participation of the data in the current version, thus we call this stage as a self-simplification process. However, the modules selected in this stage cannot guarantee to be representative for the data of the current version, which means that the distribution differences of the data between the two versions have not been addressed yet. Hence, in the second stage, we further utilize a **D**issimilarity-based **S**pare **S**ubset **S**election (**DS3**) method [27] to select a representative module subset from the prior version based on the pairwise dissimilarities between the modules of the two versions. The selected module subset can effectively represent each module of the current version, which promotes to relieve the distribution differences.

As we preserve the modules of the prior version that well represent the modules of the current version and eliminate the irrelevant modules, the classification models built on this selected subset are more targeted to the data of the current version. Since this module reduction process requires the assistance of the data in the current version to further refine the selected modules in the first stage, thus we call this stage as an auxiliary-refining process. After completing the two simplification processes, the selected module subset not only contains the important information of the original data in the prior version, but also well expresses the data of the current version. Thus, the resulting module subset is expected to achieve encouraging performance of CVDP.

4.2 The Used Methods and Proposed CVDP Framework

To deal with the representation issue among the software modules in the cross version scenario, we propose a new CVDP framework TSTSS. Subsection 4.2.1 introduces the used SMRS method during the self-simplification stage, subsection 4.2.2 describes the used DS3 method during the auxiliary-refining stage, subsection 4.2.3 gives our proposed CVDP framework TSTSS.

4.2.1 The SMRS Method

As the goal of the first stage of TSTSS is to simplify the data of prior version by only retaining part of important modules, we employ a self-representation learning method SMRS to select a module subset that minimizes the reconstruction errors without losing important information. To the best of our knowledge, there are no literatures related to the self-simplifications task in defect prediction studies.

SMRS is a modification version of dictionary learning framework aiming at finding

a sparse representation of the original data. Considering traditional dictionary learning framework usually selects some important data points which do not coincide with the actual ones in the original data, Vidal et al. [144] improved it by replacing the dictionary with the matrix of the data points. This guarantees to select the important points from the actual data space. This section details the SMRS method as follows.

We denote the modules of the prior version as $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_M] \in \mathbb{R}^{r \times M}$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, ..., x_{ir}]^{\mathrm{T}} \in \mathbb{R}^r$, M and r indicate the number of modules and features of the prior version, respectively. SMRS aims to minimize the following objective function:

$$\sum_{i=1}^{M} \|\mathbf{x}_i - \mathbf{X}\mathbf{c}_i\|_2^2 = \|\mathbf{X} - \mathbf{X}\mathbf{C}\|_F^2. \tag{4.1}$$

where $\mathbf{C} = [c_1, c_2, ..., c_M] \in \mathbb{R}^{M \times M}$ represents the coefficient matrix, $\mathbf{c}_i = [c_{i1}, c_{i2}, ..., c_{iM}] \in \mathbb{R}^M$ denotes the coefficient vector of $\mathbf{x}_i$ over $\mathbf{X}$, and $\|\cdot\|_F$ denotes the Frobenius norm (or l_2 norm). The motivation is to select a module subset whose linear combination is used to restructure the original data of the prior version. For this purpose, we add a term $\|\mathbf{C}\|_{0,q} \leq k_1$ to constrain the selected module number, where the $\|\cdot\|_{0,q}$ denotes the l_0/l_q norm and is defined as $\|\mathbf{C}\|_{0,q} = \sum_{i=1}^{M} \mathbb{I}(\|\mathbf{c}^i\|_q)$, c^i denotes the ith row of the coefficient matrix $\mathbf{C}$ and $\mathbb{I}(\cdot)$ denotes the indicator function. In addition, we add a term $\mathbf{1}^{\mathsf{T}}\mathbf{C} = \mathbf{1}^{\mathsf{T}}$ to constrain that the coefficient sum of each column in matrix $\mathbf{C}$ is equal to 1. [28]. Thus, we have the following optimization formulation:

$$\begin{aligned} \min_{\mathbf{C}} \quad & \|\mathbf{X} - \mathbf{X}\mathbf{C}\|_F^2 \\ \text{s.t.} \quad & \|\mathbf{C}\|_{0,q} \leq k_1, \mathbf{1}^{\mathsf{T}}\mathbf{C} = \mathbf{1}^{\mathsf{T}}. \end{aligned} \tag{4.2}$$

However, Eq.(4.2) is a NP-hard problem as it needs to search all the subset of the k_1 columns of $\mathbf{X}$. To alleviate this problem, an alternative way is to use some strategies to appropriately relax the harsh constraints, such as using l_1 relaxation to replace the l_0/l_q

norm with l_1/l_q norm and relaxing the positive integer k_1 to real number μ [144]. Then Eq.(4.2) is relaxed to the formulation as

$$\begin{aligned} \min_{\mathbf{C}} \quad & \|\mathbf{X} - \mathbf{X}\mathbf{C}\|_F^2 \\ \text{s.t.} \quad & \|\mathbf{C}\|_{1,q} \leq \mu, \mathbf{1}^\mathsf{T}\mathbf{C} = \mathbf{1}^\mathsf{T}. \end{aligned} \tag{4.3}$$

Using Lagrange multiplier method, Eq.(4.3) can be rewritten as:

$$\begin{aligned} \min \quad & \lambda_1\|\mathbf{C}\|_{1,q} + \frac{1}{2}\|\mathbf{X} - \mathbf{X}\mathbf{C}\|_F^2 \\ \text{s.t.} \quad & \mathbf{1}^\mathsf{T}\mathbf{C} = \mathbf{1}^\mathsf{T}, \end{aligned} \tag{4.4}$$

where λ_1 is the regularization parameter used to control the number of the selected modules. Eq.(4.4) can be solved under the **A**lternating **D**irection **M**ethod of **M**ultipliers (**ADMM**) optimization framework [32, 13].

Given a specific parameter λ_1, we will select a simplified but important module subset $\mathbf{S}$ with m modules as $\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_m] \in \mathbb{R}^{r\times m}$, where $\mathbf{s}_i = [s_{i1}, s_{i2}, ..., s_{ir}]^\mathsf{T} \in \mathbb{R}^r$.

4.2.2 The DS3 Method

As the goal of the second stage of TSTSS is to reserve the modules of the prior version that are representative to the ones in the current version, we utilize a cross-domain subset selection method DS3 [27] to pick up a more refined module subset from $\mathbf{S}$. This section details the DS3 method based on the pairwise dissimilarities between the modules of the two versions as follows.

Here, we denote the modules of the current version as $\mathbf{T} = [\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_n] \in \mathbb{R}^{q\times n}$, where $\mathbf{t}_j = [t_{j1}, t_{j2}, ..., t_{jq}]^\mathsf{T} \in \mathbb{R}^q$, n and q indicate the number of modules and features of the current version, respectively. Note that r is equal to q in our CVDP scenario. In addition,

we define a dissimilarity matrix $\mathbf{D} = [d_{ij}]$ (i=1,...,m and j=1,...,n) between the modules of the two versions, where d_{ij} indicates the dissimilarity between the module $\mathbf{s}_i$ and $\mathbf{t}_j$ which indicates how well $\mathbf{s}_i$ represents $\mathbf{t}_j$. The purpose of DS3 is to find a representative module subset of $\mathbf{S}$ that can effectively represent each module of $\mathbf{T}$.

Dissimilarity matrix $\mathbf{D}$ is formulated as

$$\mathbf{D} = \begin{bmatrix} \mathbf{d}_1^{\mathsf{T}} \\ \vdots \\ \mathbf{d}_m^{\mathsf{T}} \end{bmatrix} = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \cdots & d_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{4.5}$$

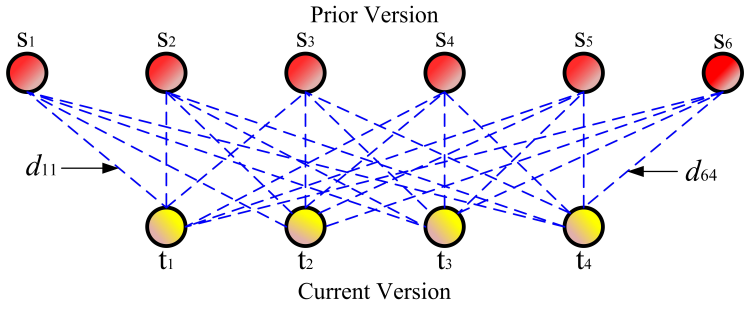
where $\mathbf{d}_i \in \mathbb{R}^n$ is the ith row of $\mathbf{D}$, i.e., the dissimilarities between the ith module of $\mathbf{S}$ and each module of $\mathbf{T}$.

To indicate whether a module in $\mathbf{S}$ is a representative of the module in $\mathbf{T}$, we define a 0-1 indicator matrix $\mathbf{P}$ as

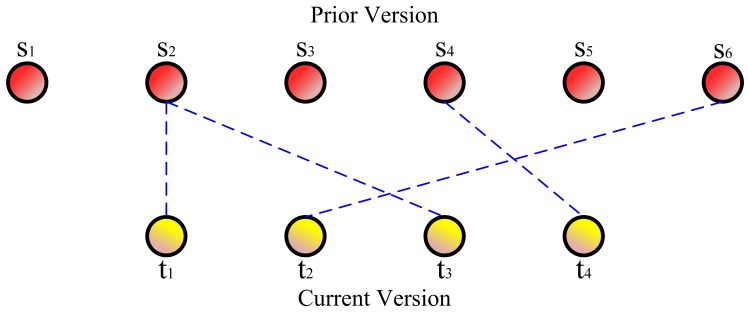
$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_1^{\mathsf{T}} \\ \vdots \\ \mathbf{p}_m^{\mathsf{T}} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{4.6}$$

where $p_{ij} \in \{0, 1\}$ is the indicator of $\mathbf{s}_i$ representing $\mathbf{t}_j$. $p_{ij} = 1$ indicates that $\mathbf{s}_i$ is a representative of $\mathbf{t}_j$, otherwise not. To guarantee that each $\mathbf{t}_j$ is represented by one $\mathbf{s}_i$, we have the constrain as $\sum_{i=1}^{m} p_{ij} = 1$.

To have a better understanding of the above description, we provide a graphic depiction in Figure 4.1. Each red circle denotes a module of the prior version while each yellow circle represents a module of the current version, and the dotted line signifies the correlation (i.e., dissimilarity in DS3) between two modules. Note that if there exists a dotted line between a red circle and a blue circle after conducting DS3 method, it means that the red circle can well represent the yellow circle. Figure 4.1(a) shows the inputs of



(a) Pairwise dissimilarities between two versions



(b) Selected modules that represent the current version

Figure 4.1: An illustration of the function of DS3.

DS3 method, i.e., the pairwise dissimilarities between the modules of the two versions. Figure 4.1(b) depicts the results after running DS3 method. It shows that DS3 method selects 3 modules (i.e., $\mathbf{s}_2$, $\mathbf{s}_4$ and $\mathbf{s}_6$) from the prior version as the representatives to represent each module of the current version. Note that there is only one dotted line connecting with each yellow circle since we assign each module of the current version with only one module of the prior version. However, there can have multiple dotted lines connecting with each red circle since each module of the prior version can represent multiple modules of the current version, for example, $\mathbf{s}_2$ can represent $\mathbf{t}_1$ and $\mathbf{t}_3$.

To select a representative subset of $\mathbf{S}$ based on the dissimilarity matrix $\mathbf{D}$, DS3 simultaneously optimizes the following two terms: minimize the representing cost towards

$\mathbf{T}$ by $\mathbf{S}$ and the number of selected modules from $\mathbf{S}$ as follows:

$$\begin{aligned} \min_{\{p_{ij}\}} \quad & \sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}p_{ij} + \lambda_2 \sum_{i=1}^{m} \mathbb{I}(\|\mathbf{p}_i\|_q) \\ \text{s.t.} \quad & \sum_{i=1}^{m} p_{ij} = 1, \forall j; \quad p_{ij} \in \{0,1\}, \forall i,j, \end{aligned} \tag{4.7}$$

where the first term refers to the representing cost towards $\mathbf{T}$ by $\mathbf{S}$, and the second term denotes the number of selected representatives from $\mathbf{S}$ which corresponds to the number of nonzero rows in matrix $\mathbf{P}$. More specifically, if $\mathbf{s}_i$ is selected as a representative of $\mathbf{t}_j$, then the representing cost towards $\mathbf{t}_j$ by $\mathbf{s}_i$ is calculated as $d_{ij}p_{ij} \in \{0, d_{ij}\}$, thus the representing cost towards $\mathbf{T}$ by $\mathbf{S}$ is $\sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}p_{ij}$. In addition, if $\mathbf{s}_i$ is a representative towards some modules of $\mathbf{T}$, then not all elements in the ith row of $\mathbf{P}$ are zeros, i.e., $\sum_{j=1}^{n} p_{ij} \neq 0$. Lower control parameter λ_2 means more candidates will be selected.

However, Eq.(4.7) is a non-convex optimization problem. Elhamifar et al. [27] suggested to replace $\sum_{i=1}^{m} \mathbb{I}(\|\mathbf{p}_i\|_q)$ with l_q norm $\|\mathbf{p}_i\|_q$ and relax the 0-1 constraint $p_{ij} \in \{0,1\}$ to continuous interval constraint $p_{ij} \in [0,1]$. Then Eq.(4.7) is relaxed to the following formulation:

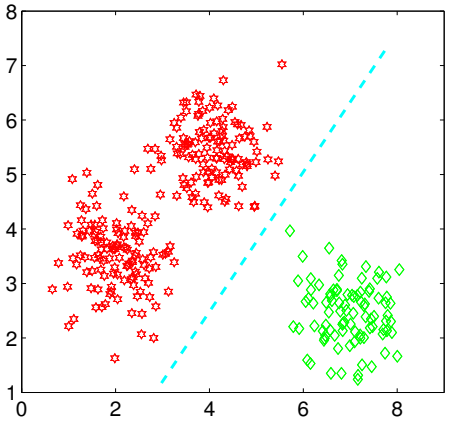
$$\begin{aligned} \min_{\{p_{ij}\}} \quad & \sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}p_{ij} + \lambda_2 \sum_{i=1}^{m} \|\mathbf{p}_i\|_q \\ \text{s.t.} \quad & \sum_{i=1}^{m} p_{ij} = 1, \forall j; \quad p_{ij} \in [0,1], \forall i,j. \end{aligned} \tag{4.8}$$

By using trace function, Eq.(4.8) is rewritten as the following matrix form:

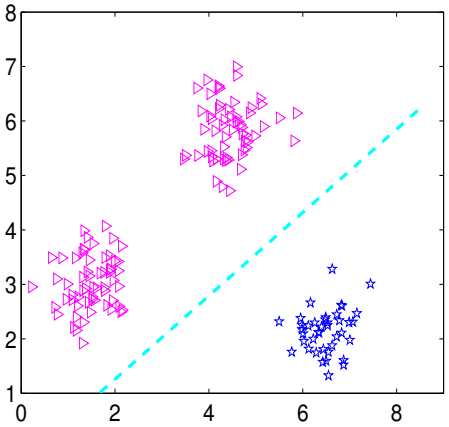
$$\begin{aligned} \min_{\mathbf{P}} \quad & \mathrm{tr}(\mathbf{D}^{\mathsf{T}}\mathbf{P}) + \lambda_2 \|\mathbf{P}\|_{1,q} \\ \text{s.t.} \quad & \mathbf{1}^{\mathsf{T}}\mathbf{P} = \mathbf{1}^{\mathsf{T}}, \mathbf{P} \in [0,1], \end{aligned} \tag{4.9}$$

where $\|\mathbf{P}\|_{1,q} = \sum_{i=1}^{m} \|\mathbf{p}_i\|_q$, $\mathbf{1}$ is a vector whose elements are all 1, and $\mathrm{tr}(\cdot)$ denotes the trace function used to calculate the sum of the diagonal elements in the matrix. Eq.(4.9) can also be solved by the ADMM framework. After obtaining the optimal solution $\widehat{\mathbf{P}}$, the line numbers of the nonzero rows in matrix $\widehat{\mathbf{P}}$ correspond to the indexes of the modules in $\mathbf{S}$ that are selected to represent the modules of $\mathbf{T}$. For example, for Figure 4.1(b) that selected 3 representative modules of the prior version (i.e., $\mathbf{S}$) to represent the 4 modules of the current version (i.e., $\mathbf{T}$), the optimal solution $\widehat{\mathbf{P}}$ is like the following formula $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$, where the values in the positions with 1 are nonzero. The line numbers of the nonzero rows in matrix $\widehat{\mathbf{P}}$ is 2, 4, 6, which indicates that the second (i.e., $\mathbf{s}_2$), the fourth (i.e., $\mathbf{s}_4$) and the sixth (i.e., $\mathbf{s}_6$) module of the prior version (i.e., $\mathbf{S}$) are selected as the representatives.

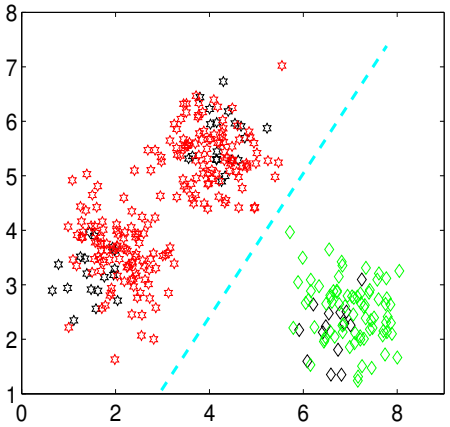
To have an intuitive feeling about the effect of DS3 for selecting the representative modules, we conduct a case study on two synthetic datasets to simulate CVDP scenario. We generate the non-defective modules of the prior version by drawing data points (red hexagrams) from a mixture of Gaussians with means (2, 3.5) and (4, 5.5) with 120 points in each set, and the defective modules by drawing 90 data points (green rhombuses) from a mixture of Gaussians with means (6.5, 2.5), as showed in Figure 4.2(a). To reflect the distribution differences between two versions, we generate the non-defective modules of the current version by drawing data points (purple triangles) from a mixture of Gaussians with means (1.5, 3) and (4.5, 6) with 60 points in each set, and the defective modules by drawing 40 data points (blue pentagrams) from a mixture of Gaussians with means (6.5, 2.5), as showed in Figure 4.2(b). Figures 4.2(c), 4.2(d), and 4.2(e) depict the selected non-defective modules (black hexagrams) and defective modules (black rhombuses) from the prior version by DS3 with 3 different λ_2 values. From the last 3 subfigures, we can see that the selected modules are close to the positions of the modules in Figure 4.2(b).



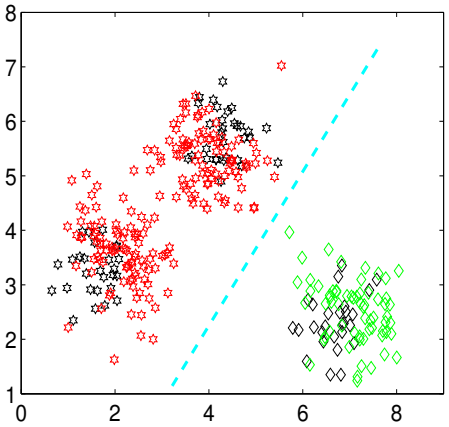
(a) Prior Version



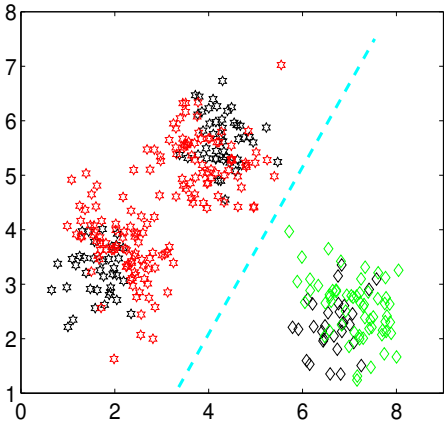
(b) Current Version



(c) $\lambda = 0.005$



(d) $\lambda = 0.001$



(e) $\lambda = 0.0001$

Figure 4.2: An example for the modules selected by DS3 with different λ_2 on synthetic data.

In addition, we observe that as the λ_2 decreases, the number of selected modules by DS3 increases.

After the second subset selection process, we obtain a more refined modules subset with m'' modules from the prior version and denote it as $\mathbf{S}' = [\mathbf{s}'_1, \mathbf{s}'_2, ..., \mathbf{s}'_{m''}] \in \mathbb{R}^{r \times m''}$, where $\mathbf{s}'_i = [s'_{i1}, s'_{i2}, ..., s'_{ir}]^\mathsf{T} \in \mathbb{R}^r$. In addition, the corresponding label set is defined as $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_{m''}] \in \mathbb{R}^{d' \times m''}$, where $\mathbf{y}_i = [y_{i1}, ..., y_{id'}]^\mathsf{T} \in \mathbb{R}^{d'}$. In this thesis, $d' = 2$ since we only have two classes of modules: non-defective and defective ones.

4.2.3 The Proposed Framework

Figure 4.3 depicts the flow chart of our proposed two-stage training subset selection method TSTSS for CVDP task. The SMRS based self-simplification stage is shown in the red rectangle and the DS3 based auxiliary-refining stage is shown in the blue rectangle. In this chapter, we use the same WELM method to construct the classification model as in Chapter 3.

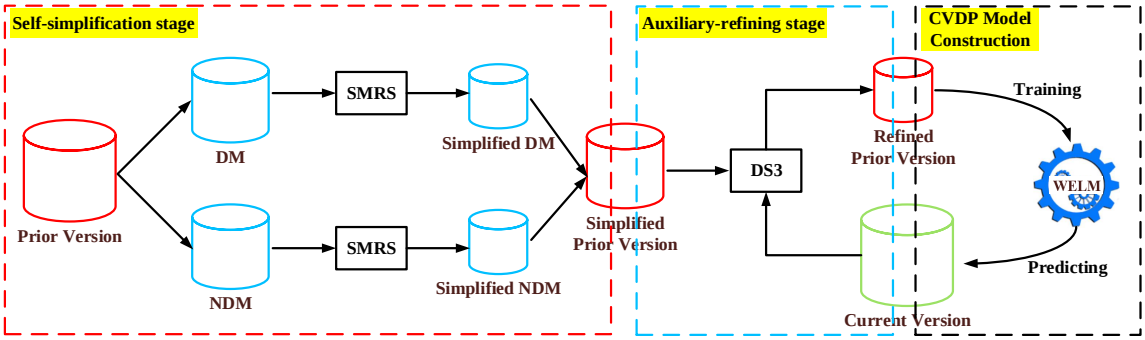


Figure 4.3: Overview of our proposed CVDP framework.

4.3 Study Setup

4.3.1 Research Questions

We empirically evaluate our devised method TSTSS by answering the following four research questions.

RQ1: How different classifiers impact the effectiveness of TSTSS on CVDP performance?

The goal of TSTSS is to select a module subset from the prior version which is used to train a defect prediction model. This question investigates whether the selected subset with WELM classifier described in Section 3.2.3 perform better than that with other classifiers for CVDP performance.

RQ2: Does our two-stage selection strategy select more effective module subset for CVDP compared with only one-stage selection strategy?

As TSTSS contains two subset selection stages: the self-simplification process with SMRS method and the auxiliary-refining process with DS3 method. This question is designed to study whether the module subset selected by TSTSS is more effective for CVDP than that selected by its 2 downgraded methods, i.e., with only one selection stage of TSTSS.

RQ3: Can TSTSS achieve better CVDP performance than other training subset selection methods?

Various subset selection methods select the module subsets based on different mechanisms. This question compares the effectiveness of TSTSS with other training subset selection methods.

RQ4: What are the differences of the modules selected by the training subset selection methods in RQ3?

Distinct training subset selection methods differ in the selected modules which lead to different CVDP performance. This question explores the differences among the module subsets selected by the methods in RQ3.

4.3.2 Benchmark Dataset

In this chapter, we use 67 versions of 17 software projects provided by Madeyski and Jureczko [87] as our benchmark dataset, called PROMISE dataset. Each software module denotes a class file of the Java project and is characterized by 20 module features with a defect label. The 20 features are calculate by a tool called CKJM and the defect label is identified by a tool called BugInfo. The original defect label is the defect number of the module. In this thesis, we transform it into a binary label by annotating the modules without defect as 0, otherwise as 1.

Detailed statistic description of each version for all 17 projects is reported in Table 4.1, where # M, # DM, % DM denote the number of modules, the number of defective modules and the defect ratio, respectively. Note that the number in the bracket of the column with # DM measures the ratio of the common defective module number across the versions to the defective module number of the prior version. Note that since the data of project Prop1, Prop2, Prop3, Prop4, Prop5, Prop42 do not contain the module name, we could not count this measure for the 6 projects. As a result, we count the percentages on total 30 cross-version pairs. From the table, we observe that the percentages on 22 out of 30 pairs are higher than 40%, which means that in most cross-version pairs, at least 40% defective modules in the prior version still remain in the next version. This observation indicates the similar uneven distribution of defects across versions. In addition, our benchmark dataset

Table 4.1: Benchmark Dataset for Cross Version Defect Prediction

Project	Version	# M	# DM	% DM	Project	Version	# M	# DM	% DM
ant	1.3	126	20	15.9%	velocity	1.4	196	147	75.0%
	1.4	178	40 (30.0%)	22.5%		1.5	214	142 (53.7%)	66.4%
	1.5	293	32 (17.5%)	10.9%		1.6.1	229	78 (40.1%)	34.1%
	1.6	352	92 (53.1%)	26.1%	xerces	init	162	77	47.5%
	1.7	745	166 (67.4)	22.3%		1.2	440	71 (41.6%)	16.1%
camel	1.0	339	13	3.8%		1.3	453	69 (23.9%)	15.2%
	1.2	608	216 (84.6%)	35.5%		1.4.4	588	437 (50.7%)	74.3%
	1.4	872	145 (44.9%)	16.6%	prop-1	9	4455	149	3.3%
	1.6	965	188 (56.6%)	19.5%		44	4081	376	9.2%
ivy	1.1	111	63	56.8%		92	3670	1287	35.1%
	1.4	241	16 (12.7%)	6.6%		128	3619	220	6.1%
	2.0	352	40 (0.0%)	11.4%		164	3541	319	9.0%
jedit	3.2.1	272	90	33.1%		192	3692	85	2.3%
	4.0	306	75 (52.2%)	24.5%	prop-2	225	1864	147	7.9%
	4.1	312	79 (65.3%)	25.3%		236	2403	76	3.2%
	4.2	367	48 (36.7%)	13.1%		245	2023	103	5.1%
	4.3	492	11 (6.3%)	2.2%		256	2025	625	30.9%
log4j	1.0	135	34	25.2%		265	2372	229	9.7%
	1.1	109	37 (64.7%)	33.9%	prop-3	285	1709	177	10.4%
	1.2	205	189 (89.2%)	92.2%		292	2330	209	9.0%
lucene	2.0	195	91	46.7%		305	2388	89	3.7%
	2.2	247	144 (71.4%)	58.3%		318	2440	365	15.0%
	2.4	340	203 (72.2%)	59.7%	prop-4	347	2906	162	5.6%
poi	1.5	237	141	59.5%		355	2802	924	33.0%
	2.0	314	37 (13.5%)	11.8%		362	2865	213	7.4%
	2.5.1	385	248 (73.0%)	64.4%	prop-5	4	3514	264	7.5%
	3.0	442	281 (77.8%)	63.6%		40	3815	466	12.2%
synapse	1.0	157	16	10.2%		85	3509	930	26.5%
	1.1	222	60 (56.3%)	27.0%		121	3445	425	12.3%
	1.2	256	86 (50.0%)	33.6%		157	2863	367	12.8%
xalan	2.4	723	110	15.2%		185	3260	268	8.2%
	2.5	803	387 (64.5%)	48.2%	prop-42	452	317	33	10.4%
	2.6	885	411 (57.1%)	46.4%		453	259	20	7.7%
	2.7	909	898 (96.8%)	98.8%		454	295	13	4.4%

contains 11 open-source projects (the first 11 projects), 5 industrial projects belonging to the insurance domain (prop1-prop5), and 1 industrial project (prop42) which is a support tool for quality assurance in software development. Table 4.2 lists the features of these projects in the PROMISE dataset.

Table 4.2: The Features of Projects from the PROMISE Dataset

1.Line of code	11.Efferent Couplings
2.Weighted Methods per Class	12.Inheritance Coupling
3.Number of Public Methods	13.Coupling Between Methods
4.Average Method Complexity	14.Lack of cohesion in methods
5.Max McCabes Cyclomatic Complexity	15.Lack of cohesion in methods
6.Avg McCabes Cyclomatic Complexity	16.Cohesion Among Methods of Class
7.Measure of Aggregation	17.Depth of Inheritance Tree
8.Coupling between object classes	18.Number of Children
9.Response for a Class	19.Measure of Functional Abstraction
10.Afferent Couplings	20.Data Access Metric

4.3.3 Evaluation Indicators

In this chapter, we use the same three traditional performance indicators (i.e., F, MCC, and AUC) and three effort-aware performance indicators (i.e., EAP, EAR, and EAF) as in Chapter 3.

4.3.4 Parameter Configuration

Regarding the implementation of the second stage of TSTSS, i.e., the DS3 method, we measure the dissimilarity with Chi-square distance following the original work [27]. The Chi-square distance of module s_i and t_j is defined as $\sum_{o=1}^{r} \frac{(s_{io}-t_{jo})^2}{2*(s_{io}+t_{jo})}$, where r denotes the feature dimension. In terms of the λ_2 in Eq.(4.9) which is used to control the final number of selected modules, a lower λ_2 value means more representative modules will be chosen. In this work, we determine the λ_2 value based on a threshold which denotes the desired proportion of the remaining modules. More specifically, we set the initial λ_2 value as 0.05 and gradually reduce it until the proportion of the selected modules towards the reversed modules in the first stage is larger than the threshold for the first time. We set 9 thresholds, i.e., 10%, 20%,..., 90%, to determine the λ_2 value. Note that we do not set threshold to 100% as it means that only the first stage SMRS is used. According to the

above description, the final proportion of the selected modules may be a little higher than the corresponding threshold.

4.3.5 Cross Version Scenario Design

In this work, we conduct the CVDP experiment between two nearest versions. The reason of this setting is that the two closest versions share more identical architectural and design characteristics, which results in a certain degree of similarity [130]. More specifically, we assume that labels of modules in a lower version is known and treat them as the training data, and assume that the labels of modules in the higher version is unknown and treat them as the test data. We do not consider where a module in a lower version is decomposed or refactorized into a few modules into the higher version. As a result, we have total 50 cross-version pairs.

4.3.6 Statistic Test Method

In this chapter, we use the same Friedman test with improved Nemenyi test as in Chapter 3 for significance analysis.

4.4 Experimental Results

4.4.1 Results for RQ1

Methods: Before studying this question, we design an experiment to determine how many modules are reserved in the first subset selection stage of our TSTSS method. Since keeping too many modules may not attain the goal of modules reduction while keeping too fewer modules may lead to excessive loss of important information and increasing the

reconstruction error, we empirically set 4 thresholds with our basic classifier WELM for comparison, including 80%, 70% 60%, 50%. As the parameter λ_1 in Eq.(4.4) is used to control the number of selected modules, we initial λ_1 as 5 and increase it with 5 at each time until the percentage of the selected modules first no less than the thresholds. As mentioned in Section 4.3.4, we set 9 percentages (from 10% to 90%) for DS3 method to control the proportion of selected modules in the second stage subset selection process. Thus, we obtain total 9 sets of results for each indicator on each cross-version pair. In this work, as we emphasize the importance and practicability of the effort-aware indicators on CVDP performance especially for EAF, we only record the results corresponding to the best EAF value among the 9 sets of results for each cross-version pair.

Results: Figure 4.4 and 4.5 show the average performance values across all cross-version pairs for TSTSS under the 4 thresholds and the statistic test results, respectively.

From Figure 4.4, we observe that TSTSS with threshold 80% always achieves the best average performance among all indicators. From Figure 4.5, we see that TSTSS with threshold 80% always obtains the best rank on all indicators. It is significantly superior to other 3 thresholds in terms of F-measure, MCC, and EAF but has no significant differences compared with EAP and EAR. Overall, 80% is the optimum threshold observed for the selected subset scale. We use this threshold for the following experiments.

To investigate the effect of different classifiers on the CVDP performance with our method TSTSS, we compare the used WELM classifier with four traditional base classifiers including k-NN, LR, CART and RF, which are commonly used in defect prediction studies [34].

Figure 4.6 and 4.7 show the average performance values across all cross-version pairs for TSTSS with the 5 classifiers and the statistic test results, respectively.

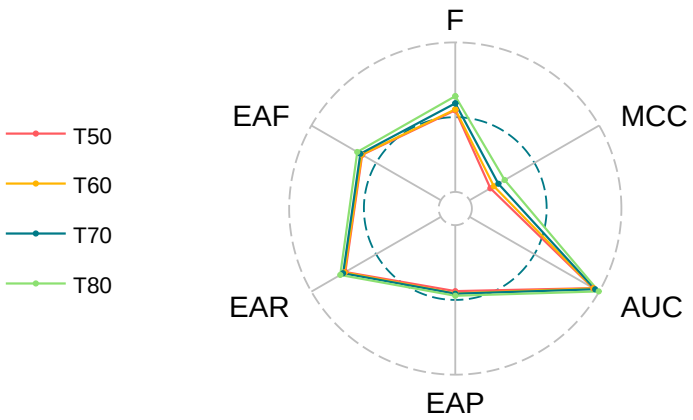


Figure 4.4: Radar charts of average indicator values for our TSTSS framework under different thresholds.

From Figure 4.6, we observe that WELM achieves the best average F-measure, EAR, and EAF, whereas RF achieves the best average MCC, and EAP. From Figure 4.7, we see that WELM belongs to the top group in terms of F-measure, AUC, EAR and EAF, whereas RF belongs to the top group in terms of MCC, AUC, and EAP. Overall, WELM and RF are the appropriate classifiers for CVDP with our proposed method TSTSS. As we emphasis

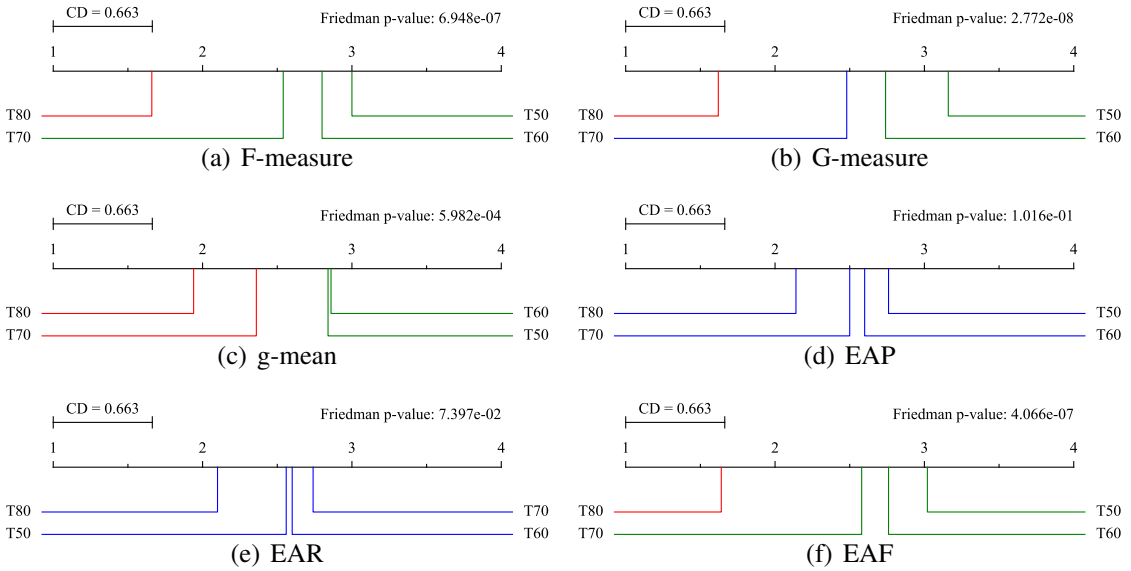


Figure 4.5: Comparison of TSTSS under different thresholds using Friedman test with Nemenyi post-hoc test in terms of six indicators.

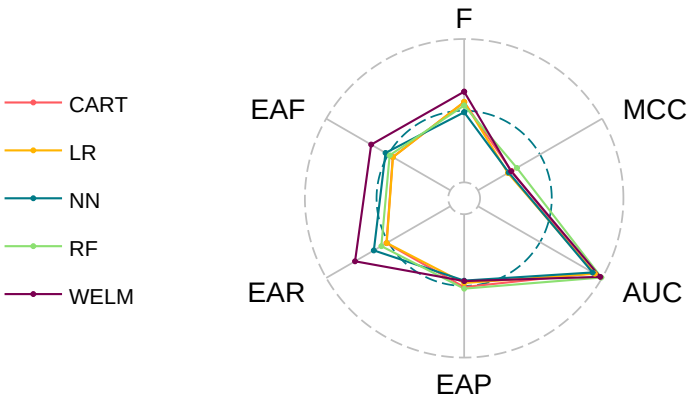


Figure 4.6: Radar charts of average indicator values for our TSTSS framework with different classifiers.

more on the importance of the effort-aware indicators in this work, thus we choose WELM as our basic classifier for the following experiments.

Answer to RQ1: Overall, our CVDP method TSTSS with WELM classifier under threshold 80% can achieve satisfactory performance, especially in terms of effort-aware indicators.

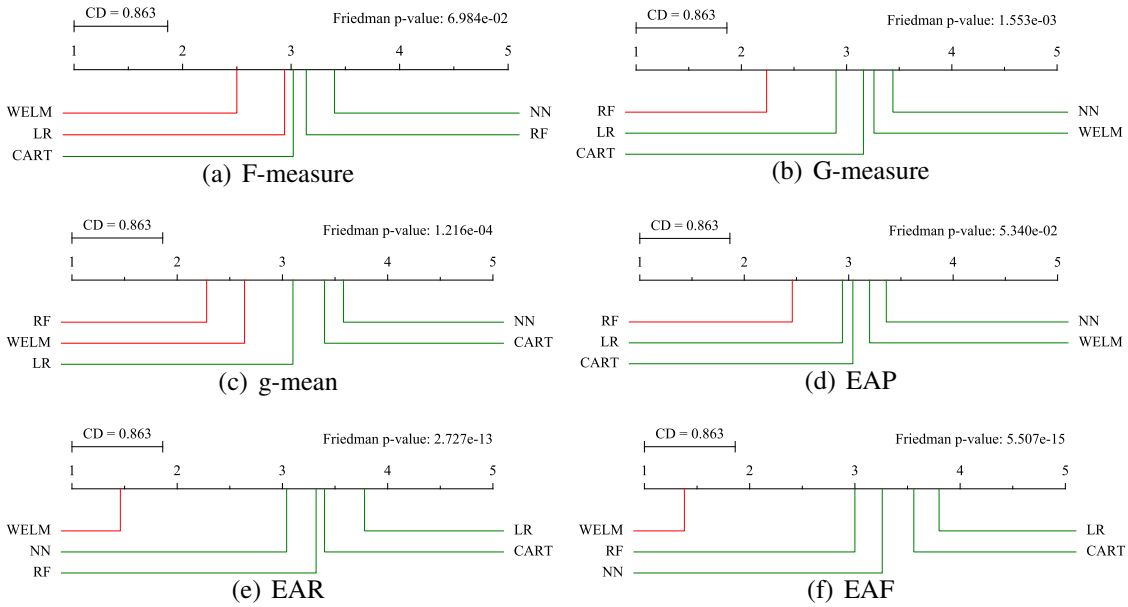


Figure 4.7: Comparison of TSTSS with different classifiers using Friedman test with Nemenyi post-hoc test in terms of six indicators.

4.4.2 Results for RQ2

Methods: To answer this question, we choose the subset selection scheme that only uses SMRS method and the scheme that only uses DS3 method as the downgraded methods for comparison. The parameter settings of the two baseline methods are the same as TSTSS for a fair comparison.

Results: Figure 4.8 shows the radar charts of the average values of each performance indicator on the cross-version pairs for each project as well as across all projects in terms of TSTSS and the two baseline methods. In the radar chart, each axis denotes a performance indicator and the value of the circle point (a.k.a vertex) on the axis corresponds to the indicator value. The vertex far away from the center has greater value. All the vertices are connected into a polygon. Each method corresponds to a polygon with a specific color. For the cross-version pairs on each project, we draw a radar chart. The values of the points on the charts are the average indicator values of these cross-version pairs. Since we have 17 different software projects, we draw 17 radar chart from Figure 4.8(a) to 4.9(q). In addition, for all the cross-version pairs, we also draw a radar chart, i.e., Figure 4.9(r). So we have total 18 radar charts in Figure 4.8.

For the ease of replicating our experimental results, we list the threshold and the corresponding λ_2 value corresponding to the best EAF in Table 4.3. From Figure 4.8 and Table 4.3, we have the following observations.

First, in terms of the three traditional indicators (i.e., F-measure (F), MCC, and AUC), from Figure 4.9(r), we observe that TSTSS achieves the best average values on the three indicators across the 50 cross-version pairs compared with the two downgraded baseline methods. More specifically, TSTSS improves the SMRS and DS3 by 39.8% and 50.4% in terms of average F-measure respectively, by 49.2% and 93.4% in terms of average MCC

respectively, and by 0.9% and 3.3% in terms of average AUC respectively.

Second, in terms of the three traditional indicators, from Figure 4.8(a), 4.8(c), 4.8(d), 4.8(f), 4.8(h), 4.9(k), 4.9(m), 4.9(n), and 4.9(q), the results show that all three average indicator values of TSTSS are higher than the two baseline methods on project ant, ivy, jedit, lucene, synapse, xerces, prop2, prop3, prop42, respectively. In addition, Figure 4.8(b), 4.8(g), 4.9(l), and 4.9(o) show that TSTSS achieves the best average F-measure and MCC than the two baseline methods on project camel, poi, prop1, and prop4, respectively; Figure 4.8(e) shows that TSTSS achieves the best average F-measure and AUC than the two baseline methods on project log4j.

Third, in terms of the three effort-aware indicators, from Figure 4.9(r), we observe that TSTSS achieves the best average values on the three indicators across the 50 cross-version pairs compared with the two baseline methods. More specifically, TSTSS improves the SMRS and DS3 by 10.4% and 13.7% in terms of average EAP respectively, by 7.0% and 7.6% in terms of average EAR respectively, and by 8.2% and 10.5% in terms of average EAF respectively.

Fourth, in terms of the three effort-aware indicators, from Figure 4.8(a), 4.8(b), 4.8(c), 4.8(d), 4.8(f), 4.8(g), 4.8(h), 4.9(m), 4.9(n), and 4.9(q), the results show that all the three average indicator values of our method TSTSS are higher than the two baseline methods on project ant, camel, ivy, jedit, lucene, poi, synapse, prop2, prop3, and prop42, respectively. In addition, Figure 4.8(e), 4.9(l), and 4.9(p) show that TSTSS achieves the better average EAR and EAF than the two baseline methods on project log4j, prop1, and prop5, respectively. Figure 4.9(k) and 4.9(o) show that TSTSS achieves the better average EAP and EAF than the two baseline methods on project xerces and prop4, respectively.

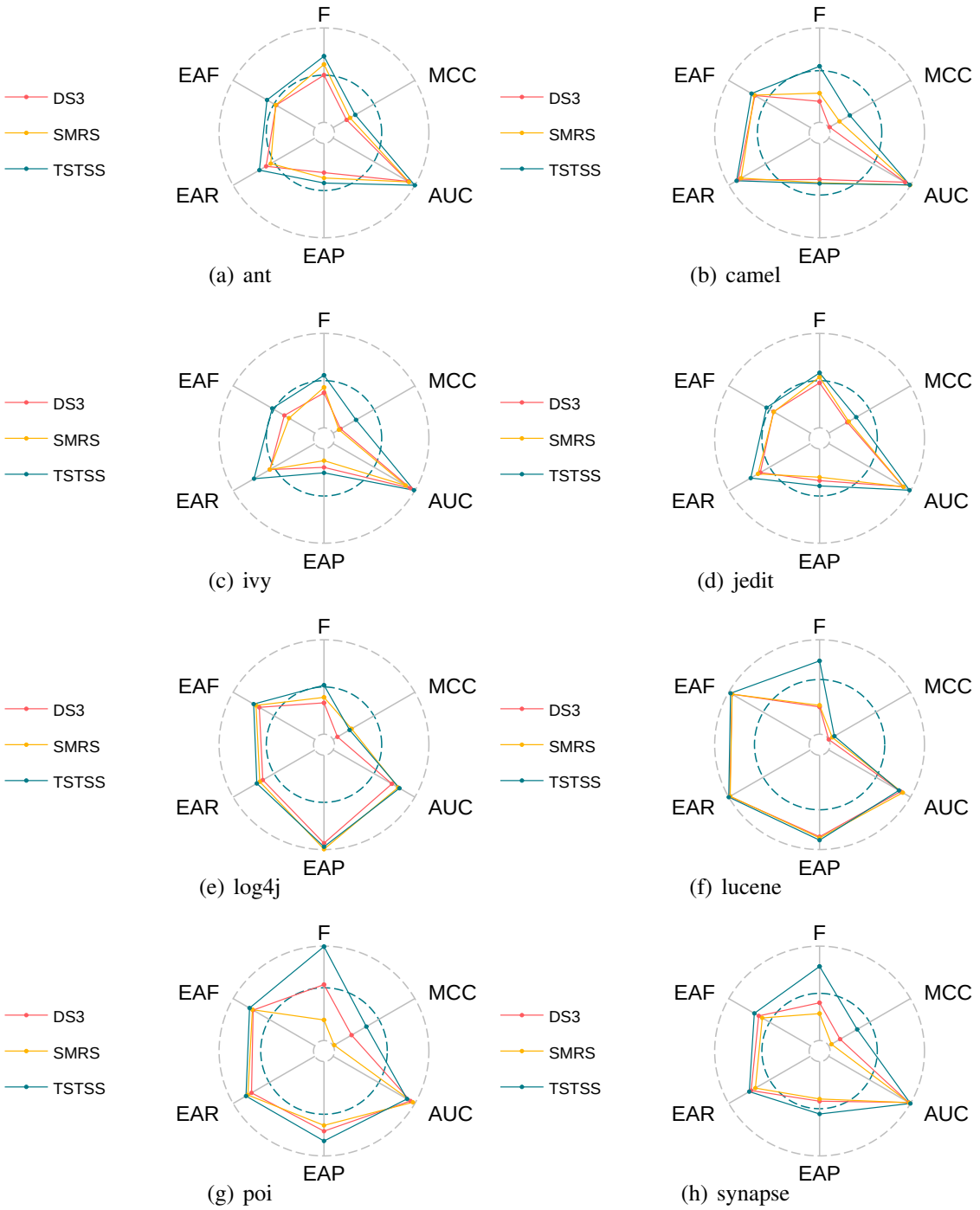


Figure 4.8: Radar charts of average values of the six indicators on the cross-version pairs of each project and across all projects in terms of TSTSS, SMRS and DS3. (Part 1)

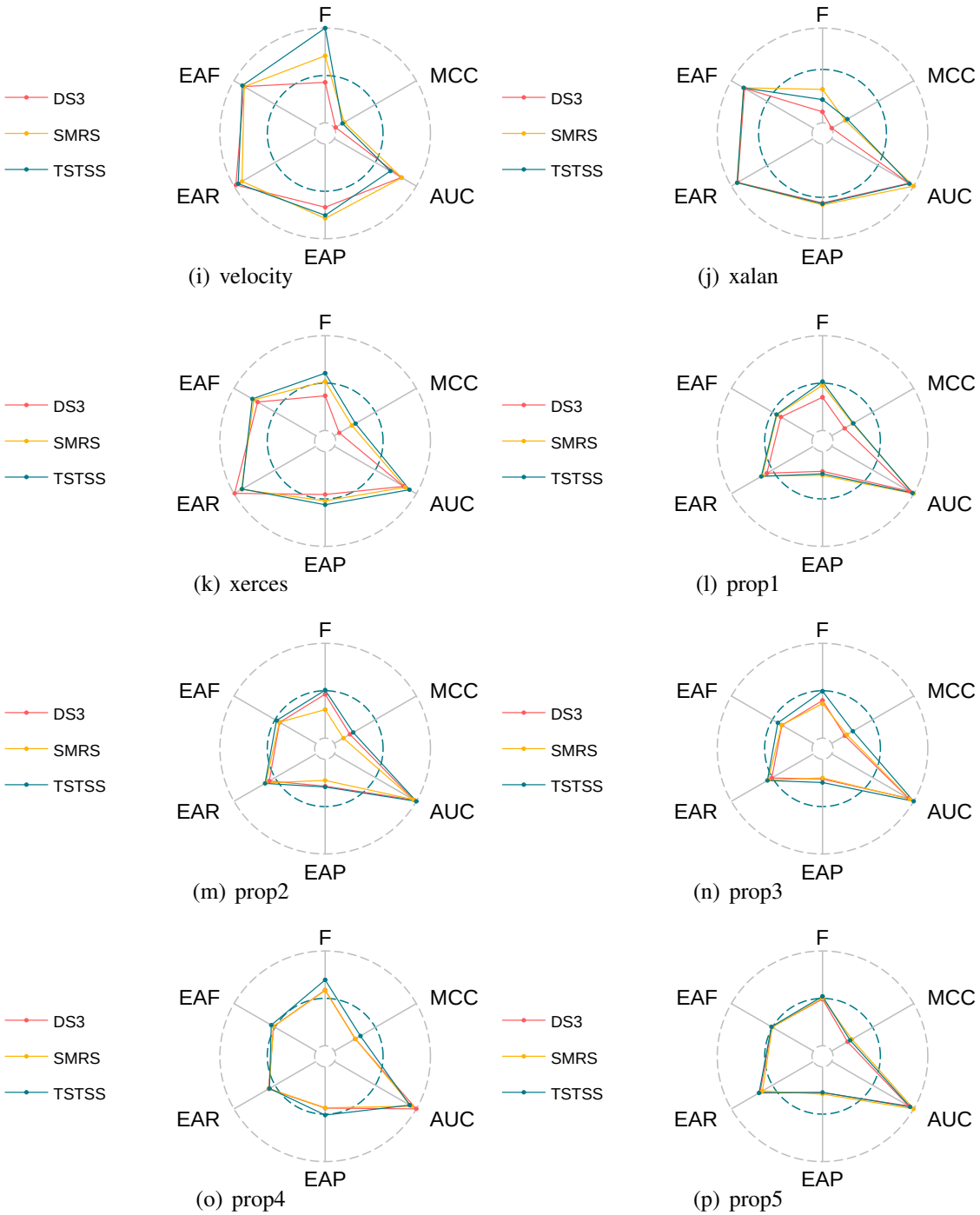
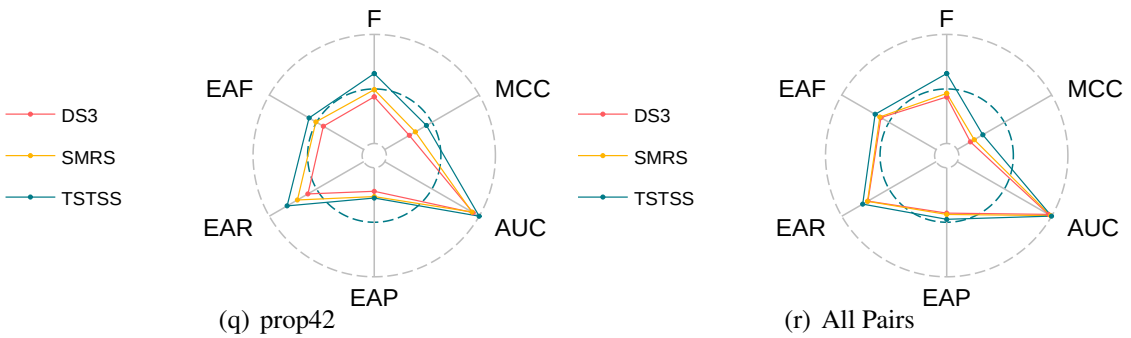


Figure 4.8: Radar charts of average values of the six indicators on the cross-version pairs of each project and across all projects in terms of TSTSS, SMRS and DS3. (Part 2)



(q) prop42 (r) All Pairs

Figure 4.8: Radar charts of average values of the six indicators on the cross-version pairs of each project and across all projects in terms of TSTSS, SMRS and DS3. (Part 3)

Fifth, in terms of the threshold in Table 4.3 which indicates the approximate percentage of the selected modules in the second stage of TSTSS, we observe that TSTSS obtains the best EAF values with less than half of original module number (corresponding to the threshold no more than 60% since we have already selected 80% of the original modules in the first stage) on 29 out of 50 cross-version pairs. It indicates that TSTSS can select only a small proportion of modules from the prior version to achieve encouraging CVDP performance on more than half of all cross-version pairs.

Sixth, Figure 4.9 visualizes the results of Friedman test and Nemenyi post-hoc test for TSTSS and the two baseline methods in terms of the six indicators. The p values (all less than 0.05) of Friedman test above the sub-figures show that there exist significant differences among the three methods on all indicators. The Nemenyi test results show that TSTSS significantly performs better than the two variant methods on all indicators.

Discussion: The takeaway lesson of the fifth finding is that when conducting CVDP, it is not necessary to use all the modules of the prior version to train the defect prediction model for the current version, since some modules have no discriminant ability or even negative impact on the classification performance. The reason why our method TSTSS outperforms SMRS is that, since SRMS selects candidates from the prior version to self-

Table 4.3: The Threshold of the Selected Module Number and the Corresponding λ_2 Value.

Cross Version Pairs			Threshold	λ_2	Cross Version Pairs			Threshold	λ_2
ant	1.3	1.4	90%	2.0E-04	velocity	1.4	1.5	60%	5.0E-04
	1.4	1.5	80%	1.0E-03		1.5	1.6	80%	3.0E-04
	1.5	1.6	40%	4.0E-03	xalan	2.4	2.5	90%	4.0E-04
	1.6	1.7	50%	3.8E-03		2.5	2.6	50%	2.4E-03
camel	1.0	1.2	50%	2.9E-03	prop1	9	44	20%	2.0E-05
	1.2	1.4	50%	1.4E-03		44	92	30%	5.0E-06
	1.4	1.6	60%	6.0E-04		92	128	30%	2.0E-05
ivy	1.1	1.4	20%	1.0E-02		128	164	20%	7.0E-05
	1.4	2.0	90%	1.0E-05		164	192	30%	1.0E-05
jedit	3.2.1	4.0	80%	3.0E-04	prop2	225	236	90%	1.0E-05
	4.0	4.1	90%	1.0E-04		236	245	60%	1.0E-04
	4.1	4.2	90%	4.0E-05		245	256	80%	1.0E-04
	4.2	4.3	80%	2.0E-04		256	265	70%	2.0E-04
log4j	1.0	1.1	50%	4.0E-03	prop3	285	292	80%	4.0E-05
	1.1	1.2	30%	1.4E-02		292	305	80%	7.0E-06
lucene	2.0	2.2	30%	1.0E-02		305	318	20%	3.6E-03
	2.2	2.4	40%	3.5E-03	prop4	347	355	70%	2.0E-05
poi	1.5	2.0	70%	4.0E-04		355	362	40%	3.0E-04
	2.0	2.5.1	90%	2.0E-04	prop5	4	40	60%	3.0E-04
	2.5.1	3.0	20%	1.4E-02		40	85	20%	2.7E-03
synapse	1.0	1.1	40%	0.006		85	121	70%	3.0E-04
	1.1	1.2	40%	4.0E-03		121	157	60%	3.0E-04
xerces	init	1.2	70%	1.3E-03		157	185	50%	7.0E-04
	1.2	1.3	50%	1.7E-03	prop42	452	453	70%	6.0E-04
	1.3	1.4.4	40%	3.4E-03		453	454	80%	1.7E-03

represent the original data without involving in the module information of the current version, it cannot guarantee that the candidates well represent the current version data. While TSTSS improves SMRS by combining a novel DS3 method to further refine the candidates with the participation of the current version data.

Since using a module subset to conduct CVDP has the potential to save memory and computing overheads to some extent compared with using the whole module set, the advantage of TSTSS will become more apparent when the scale of the defect data

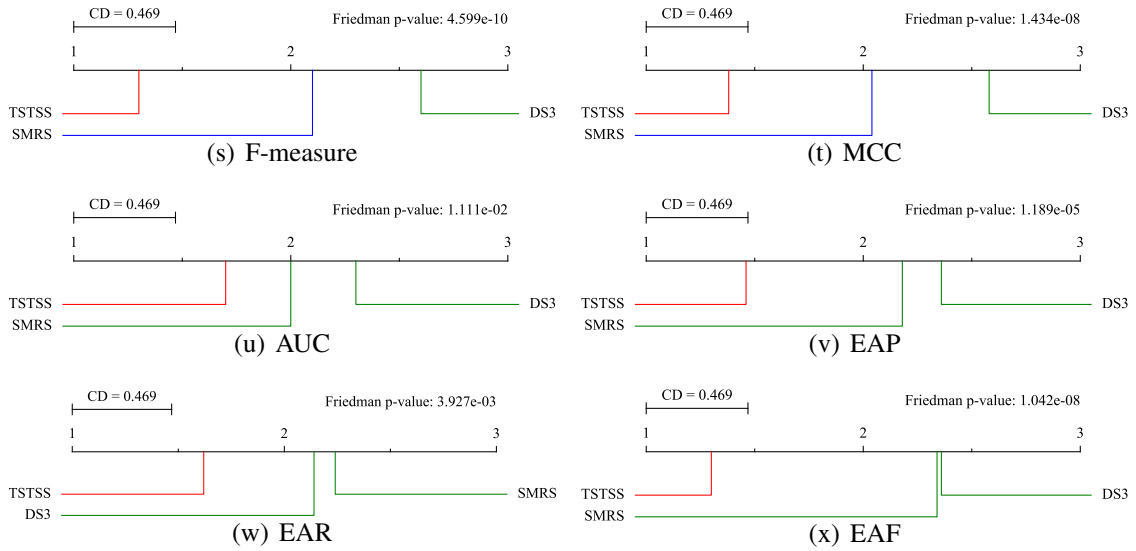


Figure 4.9: Comparison of TSTSS against SMRS and DS3 with Friedman test and Nemenyi post-hoc test in terms of all six indicators.

increases.

Answer to RQ2: To summary, TSTSS is superior to its two downgraded methods. This means that the module subset selected by the two-stage selection strategy is more effective than that selected by only one-stage selection strategy.

4.4.3 Results for RQ3

Methods: To our best knowledge, no subset selection method is tailored for CVDP. In this work, we select some typical subset selection methods that are original designed for CPDP task as our baseline methods. The modules of the prior (current) version in CVDP scenario are treated as the ones of the source (target) project in CPDP scenario. In this work, we also consider the variants of these methods as our baseline methods to enrich our experiments. The descriptions of these baseline methods are as follows:

TF1: A variant of the TF method [142] as mentioned in Section 2.2.3 which set the k as

1. This comes from the fact the this parameter is directly related to the number of selected modules, thus may affect the CVDP performance.

TF2: The original TF method [142] with $k = 10$.

PF1: The original PF method [107]. We follow the original study to set the parameter k of k-means algorithm as $\frac{M+n}{10}$ (where M and n denote the initial number of modules in the prior and current version respectively), expecting that each cluster tends to have 10 modules in average. Note that the k-means algorithm needs to randomly initialize the central points, thus, we run PF1 30 times and record the average indicator values.

PF2: A variant of PF method. The difference between PF1 and PF2 is that, for each popular module in the current version, PF1 only reserves its nearest neighbor module in the prior version, while PF2 reserves all the modules of the prior version in the clusters which contain at least one module of the current version. The setting is following the original KF method [62].

YF1: A variant of original YF method [171]. For each popular module in the current version, YF1 only select its nearest neighbor module of the prior version in the reserved clusters as PF1.

YF2: The original YF method [171]. We also follow the original study to set the cluster number as $\frac{M+n}{10}$ as the PF method.

KF1: The original KF method [62]. We follow the original study to set the two parameters, i.e., the number of minimum records and the distance, as 10 and 1, respectively.

KF2: A variant of original KF method [62] that sets the two parameters as 6 and 0.9, respectively. This comes from the fact that the performance of KF is sensitive to the

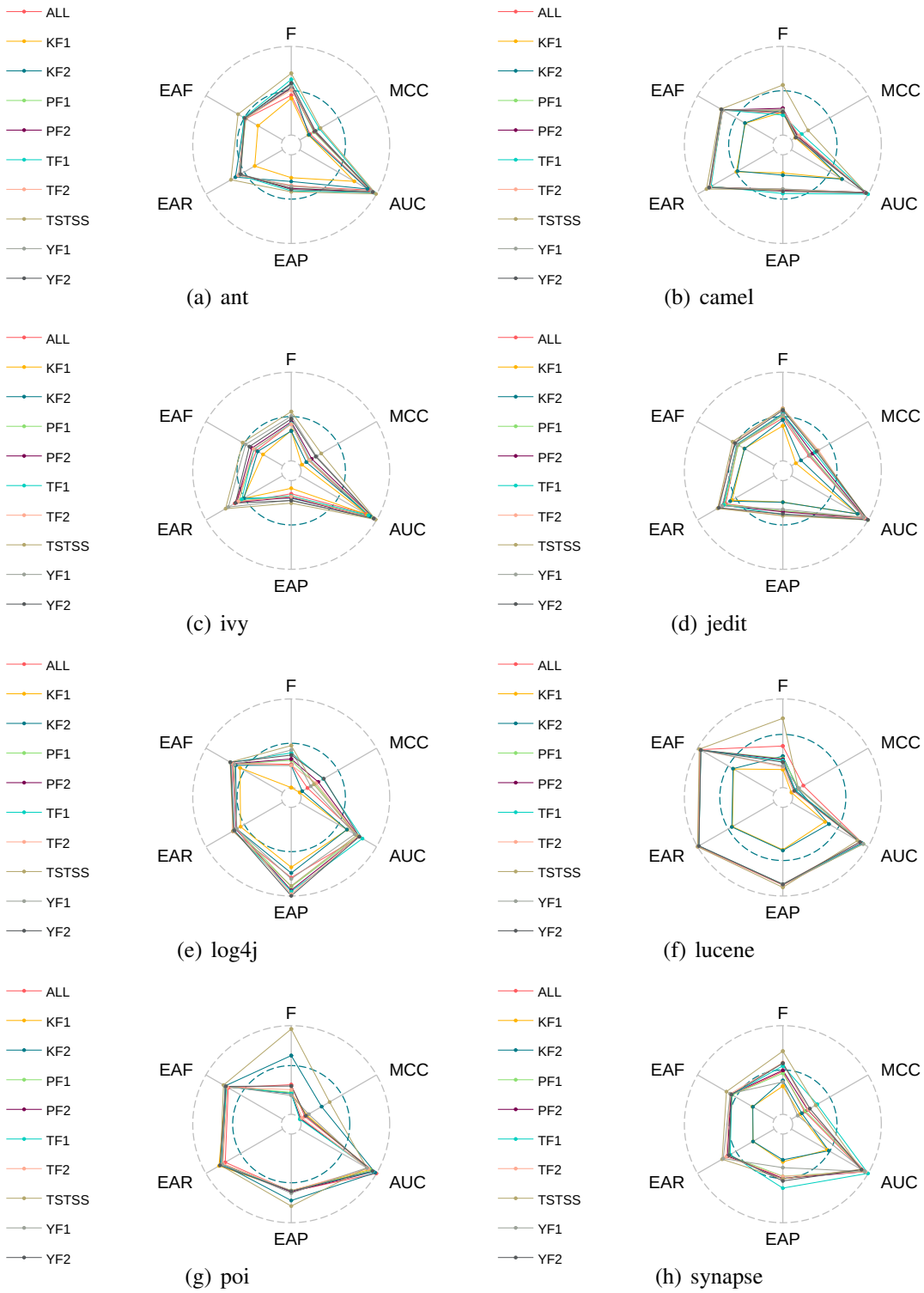
parameter setting. So we also employ the default parameter in Weka tool [12] to implement the KF2 method for comparison.

In addition, we consider the method ALL that does not perform any subset selection strategy as the most basic method for comparison.

Results: Figure 4.10 shows the radar charts of the average performance values on the cross-version pairs for each project as well as across all projects in terms of TSTSS and the nine baseline methods. From the figure, we have the following findings:

First, in terms of the three traditional indicators, from Figure 4.11(r), we find that TSTSS achieves the best average values in terms of F-measure and MCC across the 50 cross-version pairs. More specifically, average F-measure by TSTSS gains the improvement of 38.4% compared with the best average F-measure (for KF2) among the nine baseline methods; average MCC by TSTSS gains the improvement of 44.3% compared with the best average MCC (YF2) among the nine baseline methods. While average AUC by TSTSS is 0.007 lower that the best average AUC (for TF1) among the nine baseline methods.

Second, in terms of the three traditional indicators, from Figure 4.10(a), 4.10(c), 4.11(k), 4.11(m), and 4.11(n), the results show that all three average indicator values by TSTSS are better than that by the nine baseline methods on project ant, ivy, xerces, prop2, and prop3, respectively.In addition, Figure 4.10(b), 4.11(j), and 4.11(o) show that TSTSS achieves the best average F-measure and MCC than the nine baseline methods on project camel, xalan, and prop4, respectively.



(a) ant (b) camel (c) ivy (d) jedit (e) log4j (f) lucene (g) poi (h) synapse

Figure 4.10: Radar charts of average values of the six indicators on the cross-version pairs of each project and across all projects in terms of TSTSS and nine baseline methods. (Part 1)

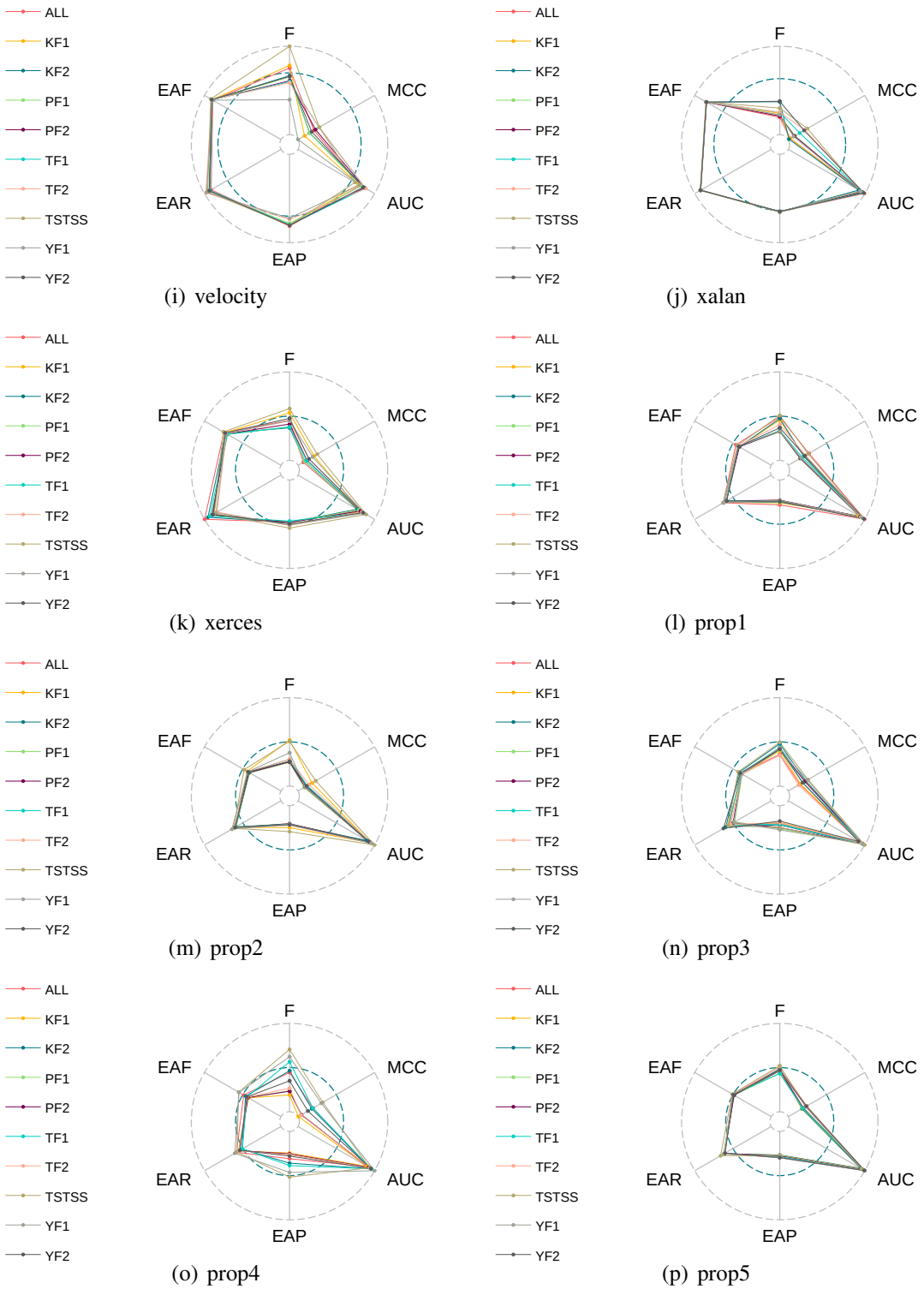
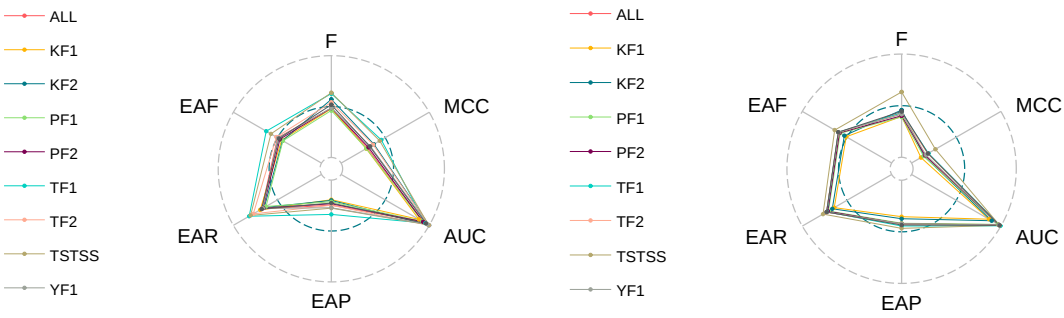


Figure 4.10: Radar charts of average values of the six indicators on the cross-version pairs of each project and across all projects in terms of TSTSS and nine baseline methods. (Part 2)



(q) prop42 (r) All Pairs

Figure 4.10: Radar charts of average values of the six indicators on the cross-version pairs of each project and across all projects in terms of TSTSS and nine baseline methods. (Part 3)

Third, in terms of the three effort-aware indicators, we also observe that TSTSS achieves the best average values on the three indicators across the 50 cross-version pairs from Figure 4.11(r). More specifically, average EAP by TSTSS gains the improvement of 5.1% compared with the best average EAP (for TF1) among the nine baseline methods; average EAR by TSTSS gains the improvement of 5.4% compared with the best average EAR (ALL, TF2 and YF1) among the nine baseline methods; average EAF by TSTSS gains the improvement of 6.9% compared with the best average EAF (TF1 and YF2) among the 9 baseline methods.

Fourth, in terms of the three effort-aware indicators, from Figure 4.10(a), 4.10(c), 4.10(d), 4.10(f), and 4.11(m), the results show that all three average effort-aware indicator values of TSTSS are higher than the nine baseline methods on project ant, ivy, jedit, lucene, and prop2, respectively. In addition, Figure 4.10(b), 4.10(e), 4.10(h), and 4.11(p) show that TSTSS achieves the best average EAR and EAF than the nine baseline methods on project camel, log4j, synapse, and prop5, respectively. Figure 4.10(g), 4.11(k), and 4.11(o) show that TSTSS achieves the best average EAP and EAF than the nine baseline methods on project poi, xerces, and prop4, respectively.

Fifth, Figure 4.11 visualizes the results of Friedman test and Nemenyi post-hoc test for the 10 methods in terms of the six indicators. The p values of Friedman test are all less than 0.05, which indicates that the differences among the 10 methods can be explained by statistically significant in terms of all six indicators. The Nemenyi test results show that TSTSS achieves the best average rank on five indicators except for AUC. TSTSS performs significantly better than the nine baseline methods in terms of F-measure, MCC, and EAF, whereas has no significant differences compared with 6, 3, 3 baseline methods in terms of AUC, EAP, and EAR, respectively.

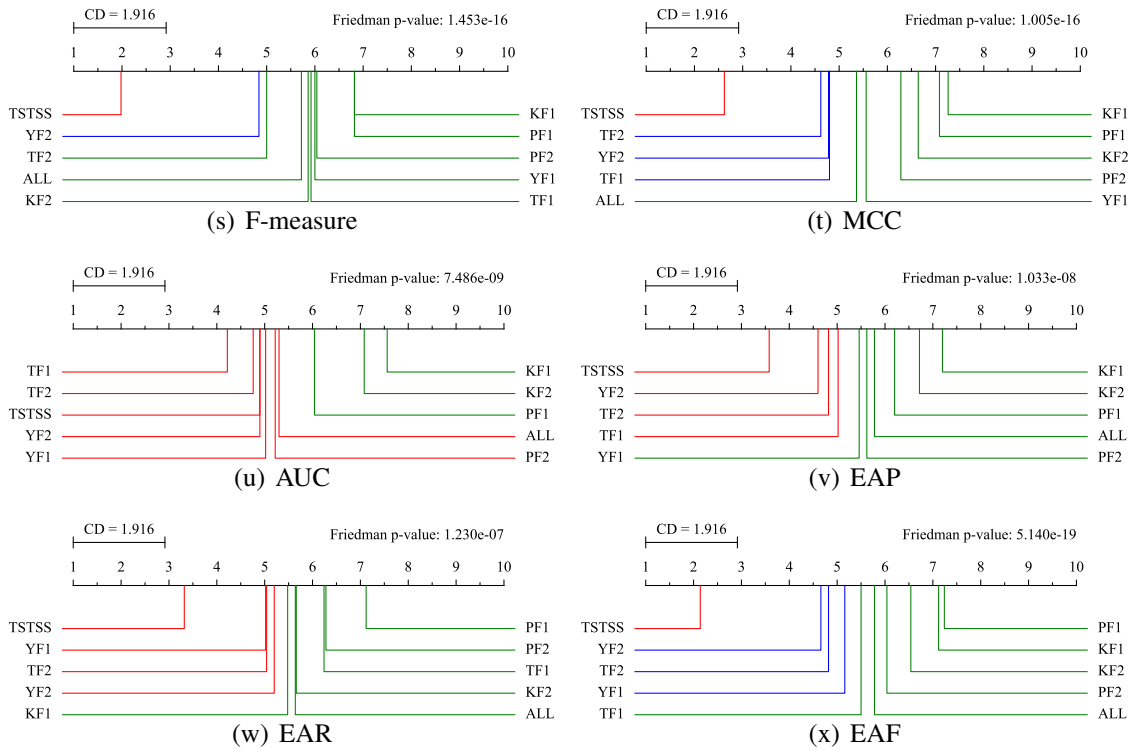


Figure 4.11: Comparison of TSTSS against nine baseline methods with Friedman test and Nemenyi post-hoc test in terms of all six indicators.

Discussion: The principles of TSTSS and the nine baseline methods are different. TSTSS selects the subset by solving an two-stage optimization problem, i.e., optimizing the reconstruction error and the representing cost, instead of simple distance based and clustering based selection as the baseline methods do. The characteristic of TSTSS is

that the selected modules are not only important to the data of the prior version but also representative to the data of the current version, therefore, the elaborately selected module subset by TSTSS is more refined and targeted. In addition, TSTSS can constrain the number of the selected modules in each stage by adjusting the corresponding parameters, while all the baseline subset selection methods cannot determine how many modules are selected. This means that TSTSS is more flexible to control the proportion of the training set according to the practice conditions, such as the constraints of the memory and computing power.

Answer to RQ3: To sum up, TSTSS is more effective to select an optimum training subset to enhance CVDP performance than other subset selection methods.

4.4.4 Results for RQ4

Methods: This question mainly focuses on two differences among these training subset selection methods. The first one is the number of selected candidates since the method that does not degrade performance with fewer selected candidate modules is desired. The second one is the defect ratios of the selected training sets by these methods since training data with higher defect ratios are helpful to train a more effective model for the identification of the defective modules. For these purposes, we record the number of selected modules by these methods and the corresponding percentages of the defective modules.

Results: Table 4.4 reports the recorded results, where # SM and % SD denote the number of the selected modules and the corresponding defect ratios. The defect ratio values are with gray shade or bold when they are higher or the same compared with the original ones, respectively. From the table, we have the following observations:

Table 4.4: Detailed Statistic of Selected Modules by TSTSS and the Baseline Methods

Cross-Version Pairs			TSTSS		TF1		TF2		PF1		PF2		YF1		YF2		KF1		KF2	
			# SM	% SD	# SM	% SD	# SM	% SD	# SM	% SD	# SM	% SD	# SM	% SD	# SM	% SD	# SM	% SD	# SM	% SD
ant	1.3	1.4	91	15.4%	116	16.4%	126	**15.9%**	111	16.5%	126	**15.9%**	115	16.5%	125	16.0%	7	0.0%	7	14.3%
	1.4	1.5	117	21.4%	143	23.8%	178	**22.5%**	137	22.9%	175	22.7%	138	23.2%	173	23.1%	31	25.8%	33	30.3%
	1.5	1.6	110	14.5%	259	9.3%	293	**10.9%**	255	9.3%	293	**10.9%**	257	9.3%	290	10.3%	88	3.4%	99	2.0%
	1.6	1.7	143	25.2%	315	26.3%	352	**26.1%**	297	25.0%	351	**26.1%**	298	25.5%	334	24.6%	118	9.3%	115	8.7%
camel	1.0	1.2	138	3.6%	270	4.1%	338	**3.8%**	241	**3.8%**	322	4.0%	255	3.1%	330	3.0%	73	0.0%	76	0.0%
	1.2	1.4	253	39.9%	473	**35.5%**	599	35.7%	440	36.5%	577	36.7%	454	35.7%	586	35.7%	198	34.8%	203	33.5%
	1.4	1.6	462	18.4%	693	16.9%	863	16.7%	656	16.2%	824	17.0%	675	15.9%	826	16.1%	355	7.6%	351	7.7%
ivy	1.1	1.4	19	68.4%	91	58.2%	111	**56.8%**	82	58.6%	111	**56.8%**	83	55.4%	105	54.3%	20	40.0%	28	39.3%
	1.4	2.0	177	6.2%	194	6.2%	241	**6.6%**	183	5.7%	241	**6.6%**	179	3.9%	231	4.8%	58	1.7%	63	1.6%
jedit	3.2.1	4.0	185	31.9%	247	32.8%	272	**33.1%**	239	32.1%	272	**33.1%**	240	31.7%	268	32.5%	74	8.1%	82	11.0%
	4.0	4.1	224	23.7%	259	22.8%	305	24.6%	252	22.5%	303	24.8%	258	23.3%	297	24.6%	88	9.1%	88	5.7%
	4.1	4.2	228	25.0%	261	26.1%	311	25.4%	254	24.7%	311	25.4%	254	23.2%	304	24.0%	76	9.2%	82	8.5%
	4.2	4.3	239	13.4%	301	13.3%	367	**13.1%**	281	13.3%	360	13.3%	290	12.4%	357	12.0%	86	2.3%	94	2.1%
log4j	1.0	1.1	56	28.5%	96	26.0%	133	25.6%	94	25.3%	135	**25.2%**	95	25.3%	133	25.6%	11	9.1%	21	4.8%
	1.1	1.2	29	27.6%	100	32.0%	109	**33.9%**	96	32.0%	109	**33.9%**	97	30.9%	107	32.7%	15	20.0%	17	11.8%
lucene	2.0	2.2	49	55.1%	175	44.6%	194	46.9%	169	43.9%	195	**46.7%**	170	43.5%	189	45.5%	0	0.0%	5	0.0%
	2.2	2.4	82	65.9%	212	56.6%	247	**58.3%**	199	55.9%	246	58.2%	202	56.4%	241	57.7%	7	42.9%	14	64.3%
poi	1.5	2.0	137	56.9%	179	58.7%	236	59.7%	171	58.4%	227	59.9%	169	56.8%	223	57.4%	105	61.0%	117	57.3%
	2.0	2.5.1	235	12.8%	269	12.3%	314	**11.8%**	262	12.6%	311	11.9%	267	11.6%	311	10.9%	174	5.7%	178	6.2%
	2.5.1	3.0	64	68.8%	283	56.9%	369	62.9%	263	54.5%	349	61.4%	275	56.0%	380	64.2%	199	73.4%	194	73.7%
synapse	1.0	1.1	56	12.5%	110	13.6%	155	10.3%	105	12.8%	151	10.6%	103	11.7%	154	10.4%	5	0.0%	8	0.0%
	1.1	1.2	91	20.9%	179	21.8%	222	**27.0%**	175	21.4%	222	**27.0%**	176	21.0%	221	27.1%	51	9.8%	59	8.5%
velocity	1.4	1.5	100	89.0%	116	82.8%	195	75.4%	113	81.7%	175	74.2%	109	84.4%	182	78.0%	37	94.6%	46	91.3%
	1.5	1.6	147	61.9%	176	64.2%	214	**66.4%**	174	64.1%	214	66.3%	175	64.0%	213	66.2%	80	43.8%	92	50.0%
xalan	2.4	2.5	532	13.5%	575	14.6%	719	**15.2%**	548	14.4%	680	15.7%	562	13.9%	702	14.0%	233	4.7%	252	5.2%
	2.5	2.6	335	55.5%	649	49.3%	801	48.3%	635	49.4%	792	47.8%	649	49.6%	795	**48.2%**	315	41.6%	314	41.1%
xerces	init	1.2	94	53.2%	111	45.9%	162	**47.5%**	96	46.1%	141	44.6%	100	45.0%	153	45.8%	41	65.9%	42	64.3%
	1.2	1.3	191	8.9%	331	14.5%	434	16.4%	326	14.1%	432	**16.1%**	328	13.7%	435	15.2%	244	13.5%	254	12.6%
	1.3	1.4.4	147	22.4%	292	17.8%	447	15.4%	268	17.4%	444	15.3%	266	16.2%	424	13.4%	234	5.1%	252	5.2%
prop1	9	44	782	10.2%	2643	4.2%	4151	3.6%	2384	4.3%	3565	3.9%	2589	3.8%	4315	2.9%	3584	1.8%	3629	1.9%
	44	92	985	16.8%	1948	11.1%	3393	9.4%	1715	11.1%	3366	8.5%	1889	10.2%	3680	9.1%	2901	6.8%	2929	7.4%
	92	128	925	**35.1%**	2515	40.7%	3518	36.3%	2500	40.6%	3651	34.9%	2503	40.6%	3633	34.9%	2818	32.2%	2907	33.3%
	128	164	580	16.0%	1568	7.3%	3300	6.5%	1350	7.0%	3245	5.3%	1430	6.2%	3286	5.1%	2652	3.4%	2653	3.3%
	164	192	933	17.7%	1687	11.1%	3205	9.2%	1464	10.1%	3142	8.2%	1527	**9.0%**	3276	7.5%	2726	5.8%	2757	5.7%
prop2	225	236	1360	8.7%	1106	9.9%	1778	8.1%	963	10.4%	1513	8.7%	1063	9.4%	1791	7.3%	1272	4.2%	1269	4.4%
	236	245	1278	4.3%	1153	4.3%	2169	3.5%	957	4.5%	1841	3.7%	1095	3.6%	2165	3.0%	1478	1.6%	1490	1.6%
	245	256	1384	6.0%	1545	5.4%	1998	5.2%	1515	5.2%	1949	5.2%	1541	5.3%	2018	5.0%	1426	3.1%	1455	3.0%
	256	265	1194	34.4%	1193	34.8%	1914	32.3%	905	33.5%	1401	33.1%	1157	34.5%	1927	30.8%	1384	30.5%	1388	31.2%
prop3	285	292	1118	11.9%	970	12.9%	1555	11.1%	755	12.9%	1191	12.3%	906	11.9%	1632	9.5%	1197	7.2%	1224	7.0%
	292	305	1503	10.5%	1496	10.5%	2146	9.7%	1305	11.3%	1817	10.9%	1468	10.3%	2287	8.7%	1602	5.4%	1634	5.4%
	305	318	385	10.4%	1764	4.1%	2247	3.9%	1699	3.8%	2141	3.9%	1757	4.0%	2372	**3.7%**	1655	1.7%	1709	1.8%
prop4	347	355	1749	5.8%	1522	5.1%	2708	**5.6%**	1246	4.9%	2321	5.2%	1431	4.4%	2635	4.9%	2148	4.2%	2190	4.5%
	355	362	927	27.4%	1904	32.7%	2699	33.6%	1812	32.7%	2583	33.9%	1892	32.4%	2731	31.5%	2160	31.3%	2214	31.1%
prop5	4	40	1847	7.3%	1992	8.9%	3302	7.8%	1682	8.9%	2759	8.0%	1884	8.3%	3335	7.1%	1965	5.6%	1960	5.6%
	40	85	630	16.8%	2445	10.9%	3653	11.2%	2398	10.5%	3714	11.6%	2416	10.5%	3720	11.7%	2395	9.8%	2458	9.5%
	85	121	1991	25.5%	2543	27.8%	3429	26.0%	2520	27.6%	3491	26.4%	2533	27.6%	3477	26.2%	2150	22.0%	2219	22.4%
	121	157	1672	17.5%	1779	17.0%	3181	13.2%	1493	17.4%	2583	14.5%	1733	16.4%	3208	12.4%	2004	7.6%	1947	7.9%
	157	185	1222	19.7%	1819	15.1%	2752	13.3%	1498	15.5%	2290	14.4%	1720	14.1%	2813	12.1%	1788	6.4%	1785	6.8%
prop42	452	453	181	11.6%	192	10.9%	292	10.3%	172	11.7%	267	10.5%	187	11.2%	283	10.2%	97	4.1%	102	4.9%
	453	454	169	**7.7%**	226	7.5%	259	**7.7%**	222	7.3%	258	**7.7%**	223	7.2%	259	**7.7%**	82	2.4%	105	1.9%

First, TSTSS selects the fewest modules on 42 out of 50 cross-version pairs compared with the first 6 baseline methods. This observation indicates that TSTSS is more effective for data reduction while still maintains the competitive CVDP performance as mentioned in 4.4.3.

Second, in terms of KF1 and KF2, they select fewer modules on 24 out of total 29 cross-

version pairs than TSTSS over the first 11 open-source projects. The reason is that they treat many modules as noises and discard them. However, they may select only one-class modules on some cross-version pairs, such as on pairs ⟨camel 1.0, camel 1.2⟩, ⟨lucene 2.0, lucene 2.2⟩, and ⟨synapse 1.0, synapse 1.1⟩. This drawback makes the classifier fail to build a discriminative CVDP model. On the contrary, TSTSS selects fewer modules than KF1 and KF2 on 18 out of total 21 cross-version pairs over the last 6 proprietary projects. As the difference between the open-source and proprietary projects is that the scale of the latter (expect for the prop42 project) is much larger than that of the former, this indicates that TSTSS is more effective to reduce the training set on large-scale defect data than KF1 and KF2.

Third, the new training set by TSTSS has higher or similar defect ratios compared with the original one on 33 out of 50 cross-version pairs, and the defect ratios of the new training sets by the first 5 baseline methods are also improved on more than half of the total cross-version pairs. While the last 3 baseline methods improve the defect ratios of the new training set on few pairs. This indicates that the training subset selection methods have greater advantages to improve the defect ratio, which is beneficial for the classification models to the identification of defective modules.

Discussion: as the baseline methods select the nearest modules directly, or first cluster the modules and then select the nearest modules in each cluster as the candidates, thus, the positions of the selected modules are nearest to the ones of the current version in the feature space. Here, we carry out an initial experiment to analyze what kind of modules are selected by our TSTSS method. Since the defect data consist of multidimensional features, we need to use a dimensional reduction technique to convert the original data into a visual plane for easy observation. For this purpose, we utilize a classic and commonly-used method **P**rincipal **C**omponent **A**nalysis (**PCA**) to transform the defect data into a two-dimensional data, and then randomly select several cross-version pairs (such as ⟨log4j-1.0,

log4j-1.1⟩, ⟨synapse-1.1, synapse-1.2⟩, and ⟨velocity-1.4, velocity-1.5⟩) for observation. First, we apply our proposed TSTSS method to select the candidates from the prior version, then use the PCA to convert the original data of two versions by maintaining 2 features (i.e., the two largest eigenvectors corresponding to the two largest eigenvalues), finally, visualize the scatter plots on a two-dimensional plane and mark the selected candidates. Figure 4.12 show the visualization results on the 3 cross-version pairs. In the figure, the blue triangle '∇' represents the module in the current version, the red cross '$+$' represents the module in the prior version, and the red cross with a red circle '$\oplus$' represents the selected candidate from the prior version. From the figure, we observe two general rules about the selected modules by TSTSS: first, TSTSS selects more modules from the prior version in the density regions; second, in sparse region, not all the nearest modules towards the current version modules will be selected, instead, TSTSS chooses a few modules from the neighbors as the representatives. This maybe the reason why TSTSS can select fewer modules from the prior version.

Answer to RQ4: From the above observations, TSTSS selects fewer modules (except compared with KF1 and KF2 on small scale defect data) with promising CVDP performance. Thus, TSTSS is a more preferred training subset selection method for CVDP task. Besides, TSTSS and the first 5 baseline methods have the potential to relieve the class imbalance issue to a certain extent by increasing the defect ratios of the selected training sets on most cross-version pairs.

4.5 Conclusion

In this chapter, we propose a two-stage training subset selection framework TSTSS for the CVDP task. First, TSTSS employs the a novel SMRS method to self-reduce the data of the previous version by retaining the software modules that play important roles

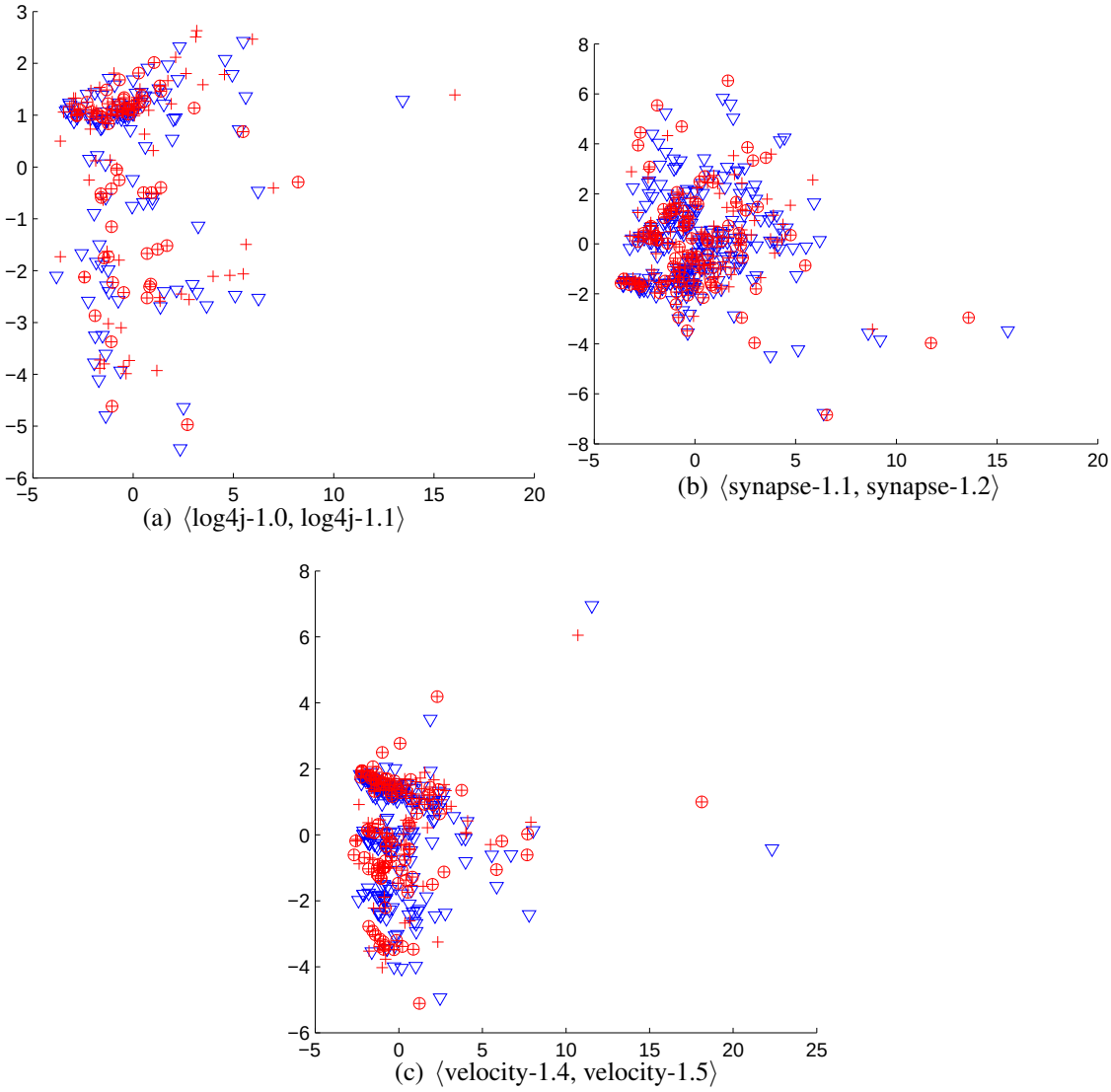


Figure 4.12: Visualization results of the selected candidates on 3 cross-version pairs.

in the reconstruction of the original data and filtering the useless modules without the participation of the software module of current version. Then, TSTSS uses the DS3 method to refine the previously selected training subset by retaining software modules that can well represent the data of the current version and filtering out software modules that have no effect on the data of the current version. We compare our proposed TSTSS framework with two variants and nine training subset selection methods on 67 versions of 17 projects from the PROMISE dataset. We use six indicators to evaluate the performance

of these methods. The experimental results show that our proposed TSTSS framework can achieve better performance in most cases.

Chapter 5

Balanced Distribution Adaptation Based Transfer Learning for Cross Project Defect Prediction

In previous two chapters, the training set and test set under IVDP and CVDP scenarios come from the same project. Thus, the two scenarios can be also called within project defect prediction which needs the labeled historical data of one project. But for the ongoing or immature projects, they maybe not historical development data to collect the labels. In this case, the work in previous two scenarios are not applicable.

Fortunately, there are publicly available labeled software defect data online whose quality has been recognized by previous researchers. Existing studies proposed to utilize the labeled data of an external project (called source project) to conduct the defect prediction task on the project (called target project) with limited or no labeled training data [103], which called CPDP. However, distinct projects have different scales and functional complexity, which may lead to the distribution differences between the data across projects. Thus, the difficulty that needs to be solved for CPDP is to eliminate the differences. Transfer learning based and training data filter based CPDP methods are

adopted to address this issue. In this work, we focus on the former one which is the mainstream of current researches.

5.1 Motivation

Transfer learning based CPDP methods transfer the knowledge of the source project to annotate the target project with the aim to minimize the distribution differences of the data between the two projects [86, 102]. There are two kinds of distribution differences, i.e., the marginal and conditional distribution differences. The former one is the distribution of the module features themselves, and the later one is the distribution of the module labels given the values of the module features. Previous transfer learning based CPDP methods, like the method in [102], mainly focus on adapting the marginal distribution difference. However, when the data of two projects are much more dissimilar, the importance of the marginal distribution is higher than that of the conditional distribution, whereas when the data of two projects are similar, the conditional distribution is more important than the marginal distribution [145]. Thus only considering one distribution is not appropriate for all cross-project pairs and will limit the CPDP performance on some cases. To overcome this deficiency, the intuition here is to simultaneously adapt the two kinds of distributions. In this work, we introduce a novel **B**alanced **D**istribution **A**daptation (**BDA**) [145] based transfer learning as our CPDP method to tackle the distribution adaptation problem. More specifically, BDA not only considers both marginal and conditional distribution differences between the data of two projects, but also assigns different importance degrees to the two kinds of distributions. Thus, it can adapt to various cross-project pairs more effectively.

5.2 The Used Methods and Proposed CPDP Framework

To deal with the data distribution difference issue in the cross project scenario, we propose to use a BDA model. Subsection 5.2.1 introduces some notation definitions, subsection 5.2.2 describes the solution process of the used BDA model, subsection 5.2.3 gives our proposed CPDP framework.

5.2.1 Notation Definitions

Assume the source project as $\boldsymbol{D_S} = \{\boldsymbol{X_S}, \boldsymbol{Y_S}\} = \{\boldsymbol{x_s^i}, \boldsymbol{y_s^i}\}_{i=1}^{\mathrm{n_s}}$, where $\boldsymbol{x_s^i}$ denotes the ith module, $\boldsymbol{X_S} \in \mathbb{R}^{\mathrm{n_s} \times \mathrm{d_s}}$ denotes the feature set of the source project, $\mathrm{d_s}$ and $\mathrm{n_s}$ separately denote the feature dimension and number of modules. In other words, the row of matrix $\boldsymbol{X_S}$ denotes the software modules and the column of matrix $\boldsymbol{X_S}$ denotes the module feature. In addition, $\boldsymbol{y_s^i}$ denotes the label of the ith module and $\boldsymbol{Y_S} \in \mathbb{R}^{\mathrm{n_s} \times 1}$ denotes the label set of the source project. Similarly, assume the target project as $\boldsymbol{D_T} = \{\boldsymbol{X_T}, \boldsymbol{Y_T}\} = \{\boldsymbol{x_t^j}, \boldsymbol{y_t^j}\}_{j=1}^{\mathrm{n_t}}$, where $\boldsymbol{x_t^j}$ denotes the jth module, $\boldsymbol{X_T} \in \mathbb{R}^{\mathrm{n_t} \times \mathrm{d_t}}$ denotes the feature set of the target project, $\mathrm{d_t}$ and $\mathrm{n_t}$ separately denote the feature dimension and number of modules. In our work, $\mathrm{d_s} = \mathrm{d_t}$, that is, the cross project data share the same feature dimension. $\boldsymbol{y_t^j}$ denotes the label of the jth module and $\boldsymbol{Y_T} \in \mathbb{R}^{\mathrm{n_t} \times 1}$ denotes the label set of the target project. Note that, the labels of the target project are unknown and need to be predicted. In addition, we define the feature space of source and target projects as $\mathcal{X}_s$ and $\mathcal{X}_t$ respectively, the label space of source and target projects as $\mathcal{Y}_s$ and $\mathcal{Y}_t$ respectively. In CPDP scenario, the defect data of the two projects have the same feature space, i.e., $\mathcal{X}_s = \mathcal{X}_t$ and the same label space, i.e., $\mathcal{Y}_s = \mathcal{Y}_t$, but have different marginal distributions, i.e., $P(\boldsymbol{x_s}) \neq P(\boldsymbol{x_t})$ and different conditional distributions, i.e., $P(\boldsymbol{y_s}|\boldsymbol{x_s}) \neq P(\boldsymbol{y_t}|\boldsymbol{x_t})$. For CPDP task, BDA adaptively minimizes the marginal distribution difference, i.e., $d(P(\boldsymbol{x_s}), P(\boldsymbol{x_t}))$, and the conditional distribution difference, i.e., $d(P(\boldsymbol{y_s}|\boldsymbol{x_s}), P(\boldsymbol{y_t}|\boldsymbol{x_t}))$

simultaneously, aiming at learning the labels $\boldsymbol{y_t}$ of the data of the target project $\boldsymbol{D_T}$ by utilizing the labeled data of the source project $\boldsymbol{D_S}$.

5.2.2 The BDA Model

The ideal transfer learning for CPDP should consider both the marginal and conditional distribution differences between the source and target projects. That is, it needs to minimize the distance between $\boldsymbol{D_S}$ and $\boldsymbol{D_T}$ as follows:

$$\begin{aligned} d(\boldsymbol{D_S}, \boldsymbol{D_T}) =& d(P(\boldsymbol{x_s}), P(\boldsymbol{x_t})) \\ &+ d(P(\boldsymbol{y_s}|\boldsymbol{x_s}), P(\boldsymbol{y_t}|\boldsymbol{x_t})). \end{aligned} \tag{5.1}$$

However, the main drawback of Eq.(5.1) is that it treats the importance of the two kinds of distributions equally. However, when the dissimilarity of the two projects is large, the margin distribution should be paid more attention, whereas when the similarity of the two projects is large, the conditional distribution should be more concerned. Therefore, for different cross project data, it is not reasonable to simply combine the two kinds of distributions with the same importance (i.e., weight). To overcome this deficiency, BDA is proposed to solve this issue by adaptively assigning different weights to the two kinds of distributions based on various cross-project pairs. BDA is formulated as follows:

$$\begin{aligned} d(\boldsymbol{D_S}, \boldsymbol{D_T}) \approx& (1-\mu) d(P(\boldsymbol{x_s}), P(\boldsymbol{x_t})) \\ &+ \mu d(P(\boldsymbol{y_s}|\boldsymbol{x_s}), P(\boldsymbol{y_t}|\boldsymbol{x_t})), \end{aligned} \tag{5.2}$$

where $\mu \in [0, 1]$ is a balance factor that is used to highlight different importance degrees of the two kinds of distributions. When the dissimilarity of the cross project data is larger, μ tends to 0, which means that the importance of the marginal distribution is emphasized; whereas when the cross project data are more similar, μ tends to 1, which means that the conditional distribution is more important. Since the balance factor μ can adaptively adjust

the importance of the two kinds of distributions for specific cross-project pair, BDA has the potential to generate a targeted training set for the CPDP task.

However, the labels of the target project are not available in advance because they are the outputs of CPDP task. In other words, $\boldsymbol{y_t}$ is unknown, which leads that it is not feasible to calculate the term $P(\boldsymbol{y_t}|\boldsymbol{x_t})$. An alternative way is to use the class conditional distribution $P(\boldsymbol{x_t}|\boldsymbol{y_t})$ to approximate the conditional distribution of the cross project data. The fact here is that when the amount of modules is adequate, the values of $P(\boldsymbol{x_t}|\boldsymbol{y_t})$ and $P(\boldsymbol{y_t}|\boldsymbol{x_t})$ are approximately equal according to the sufficient statistics [81]. To calculate the class conditional distribution, a basic classifier is built on the source project data $\boldsymbol{D_S}$ and the trained model is used to predict the labels of the target project data $\boldsymbol{D_T}$. Since the predicted labels may be not accuracy at first, multiple iterations are employed to refine the labels until the results are stable.

To calculate the discrepancy between two marginal distributions, i.e., $d(P(\boldsymbol{x_s}), P(\boldsymbol{x_t}))$, and the two conditional distributions, i.e., $d(P(\boldsymbol{y_s}|\boldsymbol{x_s}), P(\boldsymbol{y_t}|\boldsymbol{x_t}))$ in Eq.(5.2), maximum mean discrepancy (MMD) method [104] is applied to empirically estimate them. Then Eq.(5.2) is rewritten as follows:

$$\begin{aligned} d(\boldsymbol{D_S}, \boldsymbol{D_T}) = (1-\mu) & \left\| \frac{1}{\mathrm{n_s}} \sum_{i=1}^{\mathrm{n_s}} \boldsymbol{x_s^i} - \frac{1}{\mathrm{n_t}} \sum_{j=1}^{\mathrm{n_t}} \boldsymbol{x_t^j} \right\|_{\mathcal{H}}^2 \\ + \mu \sum_{c=1}^{\mathrm{C}} & \left\| \frac{1}{\mathrm{n_s^c}} \sum_{\boldsymbol{x_s^i} \in \boldsymbol{D_S^{(c)}}} \boldsymbol{x_s^i} - \frac{1}{\mathrm{n_t^c}} \sum_{\boldsymbol{x_t^j} \in \boldsymbol{D_T^{(c)}}} \boldsymbol{x_t^j} \right\|_{\mathcal{H}}^2, \end{aligned} \tag{5.3}$$

where $\mathcal{H}$ means the reproducing kernel Hilbert space, C denotes the number of different labels (C=2 in CPDP scenario), $\boldsymbol{D_S^{(c)}}$ and $\boldsymbol{D_T^{(c)}}$ denote the modules with label c in the source and target projects respectively, $\mathrm{n_s^c} = |\boldsymbol{D_S^{(c)}}|$ and $\mathrm{n_t^c} = |\boldsymbol{D_T^{(c)}}|$ denote the number of modules belonging to $\boldsymbol{D_S^{(c)}}$ and $\boldsymbol{D_T^{(c)}}$ respectively. The first term and the second term

in Eq.(5.3) represent the marginal distribution discrepancy and conditional distribution discrepancy among the cross project data, respectively.

Using the matrix tricks and regularization, Eq.(5.3) is equal to the following formula:

$$\begin{aligned} &\min \operatorname{tr}(\boldsymbol{A}^{\top}\boldsymbol{X}((1-\mu)\boldsymbol{M}_0+\mu\sum_{c=1}^{\mathrm{C}}\boldsymbol{M}_c)\boldsymbol{X}^{\top}\boldsymbol{X})+\lambda\|A\|_F^2 \\ &\text{s.t.}\boldsymbol{A}^{\top}\boldsymbol{X}\boldsymbol{H}\boldsymbol{X}^{\top}\boldsymbol{A}=\boldsymbol{I},0\leq\mu\leq 1, \end{aligned} \tag{5.4}$$

where $\boldsymbol{X}$ denotes the input data matrix that combines the feature sets of the source project $\boldsymbol{X_S}$ and the target project $\boldsymbol{X_T}$, $\boldsymbol{A}$ denotes a transformation matrix, $\boldsymbol{I}\in\mathbb{R}^{(\mathrm{n_s}+\mathrm{n_t})\times(\mathrm{n_s}+\mathrm{n_t})}$ denotes the identity matrix, and $\boldsymbol{H}=\boldsymbol{I}-(1/n)\mathbf{1}$ denotes a centering matrix. In addition, $\boldsymbol{M}_0$ and $\boldsymbol{M}_c$ are MMD matrices that can be calculated using Eq.(5.5) and Eq.(5.6) as follows:

$$(\boldsymbol{M}_0)_{ij}=\begin{cases}\frac{1}{\mathrm{n_s}^2}, & \boldsymbol{x_i},\boldsymbol{x_j}\in\boldsymbol{D_S}\\ \frac{1}{\mathrm{n_t}^2}, & \boldsymbol{x_i},\boldsymbol{x_j}\in\boldsymbol{D_T}\\ -\frac{1}{\mathrm{n_s}\mathrm{n_t}}, & \text{otherwise},\end{cases} \tag{5.5}$$

$$(\boldsymbol{M}_c)_{ij}=\begin{cases}\frac{1}{\mathrm{n_s}^{c2}}, & \boldsymbol{x_i},\boldsymbol{x_j}\in\boldsymbol{D_S}^{(c)}\\ \frac{1}{\mathrm{n_t}^{c2}}, & \boldsymbol{x_i},\boldsymbol{x_j}\in\boldsymbol{D_T}^{(c)}\\ -\frac{1}{\mathrm{n_s}^c\mathrm{n_t}^c}, & \begin{cases}\boldsymbol{x_i}\in\boldsymbol{D_S}^{(c)},\boldsymbol{x_j}\in\boldsymbol{D_T}^{(c)}\\ \boldsymbol{x_i}\in\boldsymbol{D_T}^{(c)},\boldsymbol{x_j}\in\boldsymbol{D_S}^{(c)}\end{cases}\\ 0, & \text{otherwise.}\end{cases} \tag{5.6}$$

The first term in Eq.(5.4) with balance factor μ is used to adapt the importance degrees of the marginal and conditional distributions, and the second term with parameter λ is a regularization term where $\|A\|_F^2$ is the Frobenius norm. The first constraint condition is used to ensure that the transformed data $\boldsymbol{A}^{\top}\boldsymbol{X}$ preserve the inner structure properties of the original data, and the second one constrains the value range of the balance factor μ.

To solve Eq.(5.4), we define the Lagrange multipliers as $\Phi = (\phi_1, \phi_2, ..., \phi_d)$, and then Eq. 5.4 can be rewritten as follows:

$$\begin{aligned} L =& \mathrm{tr}(\boldsymbol{A}^\top \boldsymbol{X}((1-\mu)\boldsymbol{M}_0 + \mu \sum_{c=1}^{\mathrm{C}} \boldsymbol{M}_c)\boldsymbol{X}^\top \boldsymbol{X}) \\ &+ \lambda \|\boldsymbol{A}\|_F^2 + \mathrm{tr}((\boldsymbol{I} - \boldsymbol{A}^\top \boldsymbol{X}\boldsymbol{H}\boldsymbol{X}^\top \boldsymbol{A})\Phi). \end{aligned} \tag{5.7}$$

We set the first-order derivative of Eq.(5.7) in terms of $\boldsymbol{A}$ as 0, i.e., $\partial L / \partial \boldsymbol{A} = 0$, and the optimization is transformed into a generalized eigendecomposition problem as follows:

$$\begin{aligned} &(\boldsymbol{X}((1-\mu)\boldsymbol{M}_0 + \mu \sum_{c=1}^{\mathrm{C}} \boldsymbol{M}_c)\boldsymbol{X}^\top + \lambda \boldsymbol{I})\boldsymbol{A} \\ &= \boldsymbol{X}\boldsymbol{H}\boldsymbol{X}^\top \boldsymbol{A}\Phi. \end{aligned} \tag{5.8}$$

As a result, the optimum transformation matrix $\boldsymbol{A}$ is obtained as the solution of Eq.(5.8). Given a threshold of feature dimension that we want to preserve for the new feature set, we can get the transformed data of the source and the target projects.

We provide an example to show the feature transformation effect of the BDA method using simulated data. For the source project, we generate 120 non-defective software modules (red circles) from a mixture of Gaussian with means (1.5, 4), and 50 defective software modules (blue circles) from a mixture of Gaussian with means (6, 3), as showed in Figure 5.1(a). To reflect the distribution differences across projects, for the target project, we generate 120 non-defective software modules (red pentagrams) from a mixture of Gaussian with means (3.5, 5), and 60 defective software modules (blue pentagrams) from a mixture of Gaussian with means (5.5, 1), as showed in Figure 5.1(b). Figures 5.1(c) depicts the mapped data of two projects by BDA method with the equal weight in the common feature space. From Figure 5.1, 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.

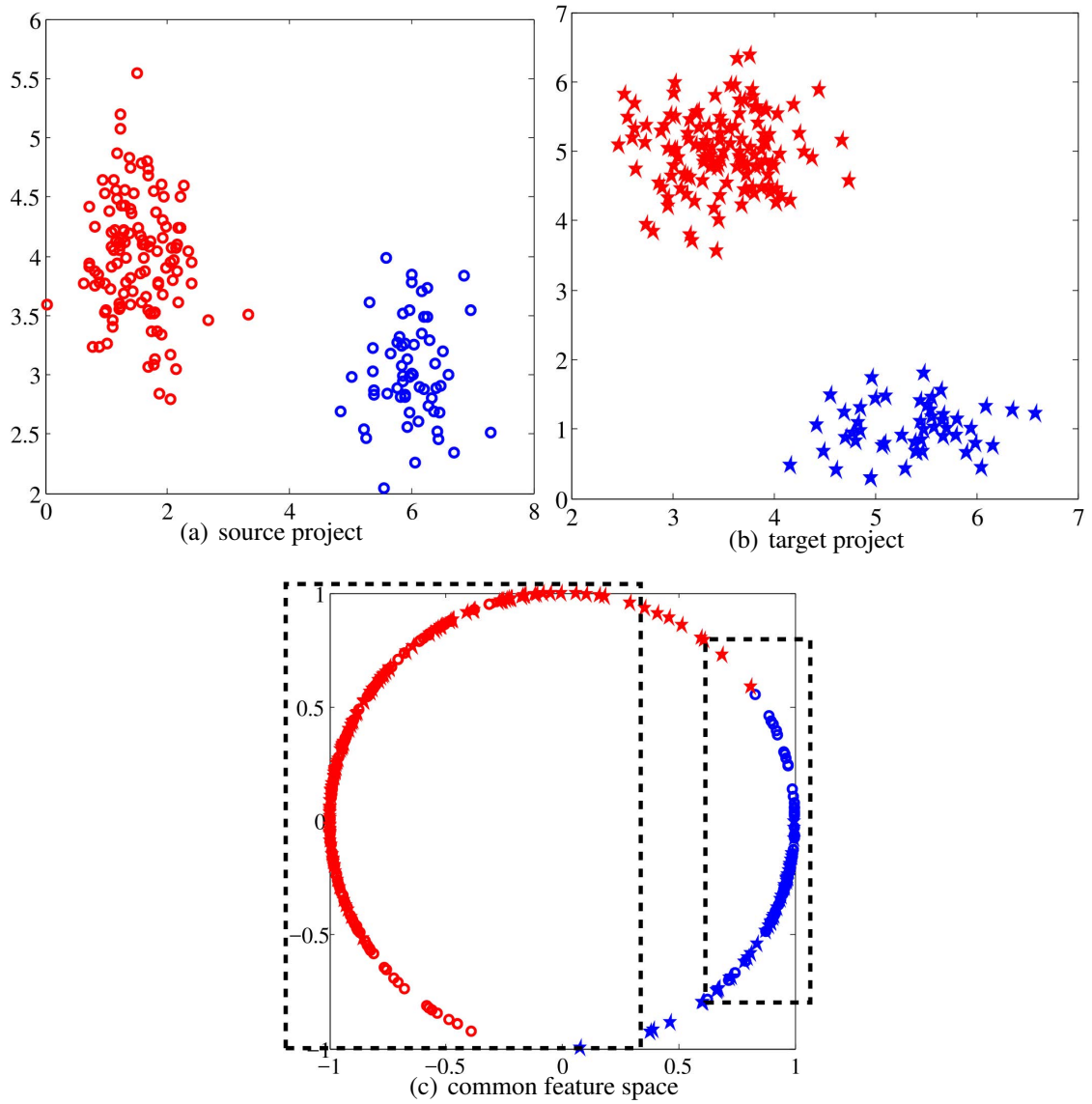


Figure 5.1: An example of the feature transformation effect by BDA.

5.2.3 The Proposed Framework

Figure 5.2 depicts the flow chart of our proposed BDA model based CPDP framework. We first use the z-score method to normalize the source project and target project individually.

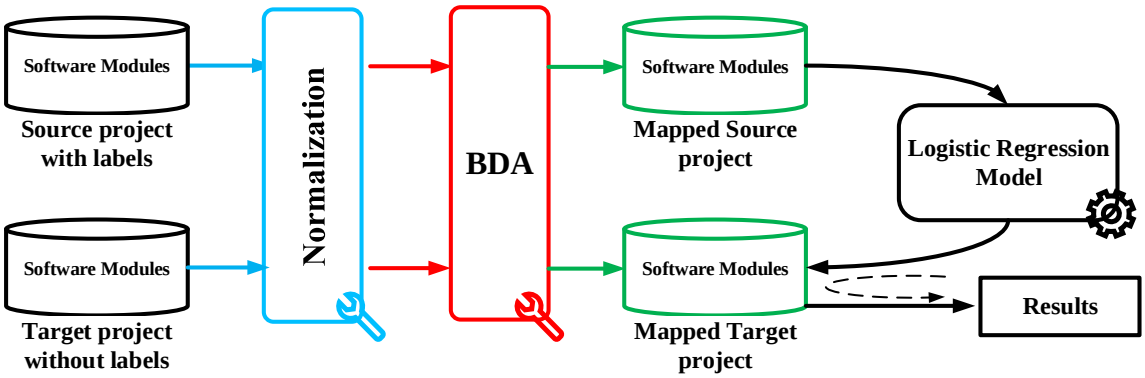


Figure 5.2: Overview of our Proposed CPDP framework.

Then, we use the BDA model to map the two project data into a common feature space. As we mainly focus on investigate the predictive effect of cross-project data mapped by the BDA model and do not consider the impact of data class imbalance, in this chapter, we use the **L**ogistic **R**egression (**LR**) classifier for classification model construction and prediction. The training process is to learn the weight vector associated to the module features and the bias parameter.

5.3 Study Setup

5.3.1 Research Questions

We empirically evaluate the BDA model by answering the following six research questions.

RQ1: How do different data normalization strategies impact the BDA performance?

Nam et al. [102] have stated that different data normalization methods can impact the performance of the CPDP model. This question is designed to investigate whether our BDA based CPDP method is affected by different data normalization methods and to find

the most suitable one for our data preprocessing.

RQ2: Is BDA more effective than the training data filter based CPDP methods?

Training data filter based CPDP methods alleviate the data distribution differences of two projects by selecting some modules from the source project that are representative to the modules of the target project. Such methods do not change the feature spaces of the two projects. However, some of the discarded modules may contain important information to distinguish the modules of different classes. Different from this kind of methods, BDA just transfers the feature spaces while reserves all modules of the source project avoiding the information loss. This question is designed to investigate whether BDA is superior to the training data filter based methods for CPDP performance improvement.

RQ3: Does BDA perform better than the transfer learning based CPDP methods?

The distribution differences of the cross project data come from two aspects: the marginal and the conditional distribution differences. In addition, according to the distinct similarity levels between the data of the two projects, the importance degrees of the two kinds of distributions vary. However, existing transfer learning based CPDP methods neither simultaneously consider the two kinds of distributions, nor focus on their different importance degrees. This question is designed to investigate whether the method considering both distributions and their importance degrees (i.e., BDA) will further improve the CPDP performance compared with other transfer learning methods.

RQ4: How different parameter settings of the selected feature dimensions impact the performance of BDA model based CPDP?

After transforming the original data of two projects into a new feature space, we empirically select 5% of original feature dimensions to carry on our experiments. This question discusses the impacts of different feature dimensions on the CPDP performance

of BDA.

RQ5: How different parameter λ values impact the performance of BDA model based CPDP?

As we set the regularization parameter λ value to 0.1 without any prior knowledge, this question discusses the impacts of different parameter λ values on the CPDP performance of BDA.

RQ6: How different classifiers impact the performance of BDA model based CPDP?

As we choose LR classifier as our basic classification model, this question discusses the impacts of different classifiers on the CPDP performance of BDA.

5.3.2 Benchmark Dataset

In this chapter, we use the NASA datasete and AEEEM dataset the same as in Chapter 3 as our benchmark dataset. Since the CPDP experiments need that the projects in the same dataset have the same feature set, we select all projects in the AEEEM dataset and five projects with 37 features in the NASA dataset to conduct our experiments. Table 5.1 reports the detailed statistic description of these projects.

5.3.3 Evaluation Indicators

In this chapter, we use the same three traditional performance indicators (i.e., F, MCC, and AUC) and three effort-aware performance indicators (i.e., EAP, EAR, and EAF) as in Chapter 3 and Chapter 4.

Table 5.1: Benchmark Datasets for Within Project Defect Prediction

Dataset	Project	# F	# M	# DM	% DM
NASA	CM1	37	327	42	12.84%
	MW1	37	251	25	9.96%
	PC1	37	696	55	7.90%
	PC3	37	1073	132	12.30%
	PC4	37	1276	176	13.79%
AEEEM	Equinox	61	324	129	39.81%
	JDT	61	997	206	20.66%
	Lucene	61	691	64	9.26%
	Mylyn	61	1862	245	13.16%
	PDE	61	1497	209	13.96%

5.3.4 Parameter Configuration

In the BDA model, there are 4 parameters that need to be specified. For the regularization parameter λ in Eq.(4.8), we set it to 0.1. As mentioned in Subsection 5.2.2, we use multiple iterations to refine the predicted labels. In our experiment, we set the maximal iterations as 10. For the balance factor μ, it is a project-specific parameter, which means that the μ value varies for different cross-project pairs. In other words, the μ value is estimated based on the data distributions of the two projects. Unfortunately, there is no effective way for such estimation [145]. In real application scenario, it is feasible to use the cross-validation strategy to determine the optimum μ value. Since this work just makes an initial exploration to investigate whether considering both two kinds of distributions with different weights can further improve CPDP performance, we set 11 different μ values, i.e., 0, 0.1,..., 0.9, 1, and run BDA on each μ value to search the optimum value. For the desired feature dimensions of the transformed source and target projects, in this chapter, we just choose to reserve 5% of total feature number.

5.3.5 Cross Project Scenario Design

In this work, we conduct the CPDP experiment on the projects in the same dataset and only consider the one-to-one cross project prediction. For example, if the EQ project in AEEEM dataset is treated as the target project, then the other 4 projects in AEEEM dataset take turns as training sets. Thus, we have a total of 20 cross-project pairs on the AEEEM dataset. Similarly, we have a total of 20 cross-project pairs on the NASA dataset. As a result, we have a total of 40 cross-project pairs in this chapter.

5.3.6 Statistic Test Method

In this chapter, we use the same Friedman test with improved Nemenyi test as in Chapter 3 and Chapter 4 for significance analysis.

5.4 Experimental Results

5.4.1 Results for RQ1

Method: In this chapter, we employ a total of six normalization techniques in [102] to answer this question. These techniques belong to two types of data normalization schemes, 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 [157]. First, we briefly describe the six normalization techniques as follows:

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

N1: Applying min-max normalization to the defect data of each project.

N2: Applying z-score normalization to the defect data of each project.

N3: Applying z-score normalization to the defect data of each project with the mean value and standard deviation from the source project. In other words, we normalize the i-th feature in both projects using the mean value and standard deviation of the i-th feature in the source project.

N4: Applying z-score normalization to the defect data of each project with the mean value and standard deviation from the target project. In other words, we normalize the i-th feature in both projects using the mean value and standard deviation of the i-th feature in the target project.

NAS:**N**ormalization **A**daption **S**election utilizes a heuristic strategy to select the optimum normalization option from the above five techniques. This heuristic strategy is based on the elements of a data characteristic vector which measures the similarity of the data characteristic between two projects [102].

Results: Table 5.2 presents the average indicator values of the BDA method with six different data normalization techniques on the cross project pairs of NASA dataset, AEEEM dataset, and across the two datasets. From this table, we have the following observations.

BDA combining two data normalization techniques, i.e., N0 and N1, achieves the worst performance. In other words, using the original feature values without any normalization

Table 5.2: Average Indicator Values of the BDA Method with Six Different Data Normalization Techniques on Each Dataset and Across All Dataset

Indicator	Dataset	BDA_N0	BDA_N1	BDA_N2	BDA_N3	BDA_N4	BDA_NAS
F	NASA	0.000	0.026	0.489	0.485	**0.496**	0.490
	AEEEM	0.142	0.172	**0.541**	0.517	0.536	0.500
	TOTAL	0.071	0.099	0.515	0.501	**0.516**	0.495
MCC	NASA	0.000	0.013	0.242	0.242	**0.251**	0.245
	AEEEM	0.060	0.069	**0.317**	0.303	0.308	0.276
	TOTAL	0.030	0.041	**0.280**	0.273	0.279	0.261
AUC	NASA	0.558	0.630	0.722	0.715	**0.729**	0.723
	AEEEM	0.672	0.670	**0.746**	0.729	0.744	0.732
	TOTAL	0.615	0.650	0.734	0.722	**0.736**	0.728
EAP	NASA	0.050	0.059	0.140	0.149	0.144	0.145
	AEEEM	0.146	0.160	**0.310**	0.300	0.299	0.280
	TOTAL	0.098	0.109	**0.225**	0.224	0.222	0.213
EAR	NASA	0.222	0.225	0.305	0.327	0.324	**0.318**
	AEEEM	0.410	0.419	0.404	0.387	0.406	**0.416**
	TOTAL	0.316	0.322	0.355	0.357	0.365	**0.367**
EAF	NASA	0.130	0.137	0.245	0.261	**0.256**	0.254
	AEEEM	0.282	0.300	0.359	0.343	0.358	**0.361**
	TOTAL	0.206	0.218	0.302	0.302	0.307	**0.308**

and using the maximum-minimum method to deal with the original feature values are not helpful for our BDA model-based CPDP framework to achieve the good performance. In addition, on all cross-project pairs from the NASA dataset, the average values of F, MCC, AUC and EAP obtained by the BDA model with four normalized techniques, i.e., N2, N3, N4 and NAS have very small differences. On all cross-project pairs from the AEEEM dataset, the average values of F, MCC, AUC and EAP obtained by the BDA model with the normalization technique N2 are much higher than those obtained by the BDA model with the normalization techniques N3, N4 and NAS. On all cross-project pairs from the two data sets, the average values of the six indicators obtained by the BDA model with the four normalized techniques, i.e., N2, N3, N4, and NAS, have small differences. Based

on the above observations, on the whole, the performance values obtained by the BDA model with the normalization technique N2 are no worse than those obtained by the BDA model with the other three normalization techniques (except for N0 and N1). Considering that (1) the normalization technique NAS needs to use a series of complex rules to choose different normalization schemes for different cross-project pairs; (2) the BDA model with the normalization technique NAS does not obviously improve the performance of the cross-project defect prediction; (3) the normalization technique N2 is commonly used in the existing defect prediction researches, we adopt the normalization technique N2 to preprocess the defect data in the following research question in this chapter.

Answer to RQ1: Our proposed BDA model based CPDP framework is not suitable to combine the raw defect data without normalization and the defect data preprocessed by the maximum-minimum normalization technique. In addition, the performance of the BDA model with different z-score normalization strategies and the normalization adaption selection (NAS) strategy has small differences on the whole.

5.4.2 Results for RQ2

Methods: To answer this question, we employ some training data filter based CPDP models as our baseline methods including ALL, NN-Filter [142], Peter-Filter [107], Yu-Filter [171], and HISNN [119]. ALL means that we use all the modules of the source project to train the classification model without any data filter process. We treat this method as a special data filter method and the most basic setting for comparison. HISNN method is a nearest-neighbor based hybrid training data selection method. This method uses a k-nearest neighbor to learn the local knowledge and employ Naive Bayes to learn the global knowledge. Note that HISNN method uses a hybrid rule to determine the module labels of the target project, we could not calculate the AUC indicator without the output

probability. We also implement the Kawata-Filter method with the parameter setting in the original paper and try some other settings. However, this method identifies many modules as the noise and discards them on majority cross-project pairs, which makes it impossible for us to get modules from the source project to form the training set in most cases. Thus we do not choose this method for comparison. Note that, NN-Filter, Peter-Filter, and Yu-Filter correspond to the TF2, PF1, and YF2 in Chapter 4. In order to distinguish whether these methods are used in the CVDP scenario or CPDP scenario, we give them different names. The first baseline method is a very classic CPDP method and the other three baseline methods are also specifically designed for CPDP task. This implies that the selected baseline methods are representative to some extent.

Table 5.3: Average Indicator Values of BDA Model and five Training Data Filter Methods on Each Dataset and Across All Datasets

Indicator	Dataset	ALL	NN-Filter	Peter-Filter	Yu-Filter	HISNN	BDA
F	NASA	0.391	0.419	0.315	0.389	0.213	**0.489**
	AEEEM	0.409	0.424	0.408	0.448	0.267	**0.541**
	TOTAL	0.400	0.422	0.362	0.419	0.240	**0.515**
MCC	NASA	0.152	0.187	0.065	0.192	0.044	**0.242**
	AEEEM	0.215	0.218	0.163	0.220	0.169	**0.317**
	TOTAL	0.184	0.203	0.114	0.206	0.107	**0.280**
AUC	NASA	0.645	0.686	0.563	0.687	0.000	**0.722**
	AEEEM	0.668	0.663	0.611	0.686	0.000	**0.746**
	TOTAL	0.657	0.675	0.587	0.687	0.000	**0.734**
EAP	NASA	0.131	0.134	0.099	**0.167**	0.101	0.140
	AEEEM	0.284	0.281	0.242	0.275	**0.344**	0.310
	TOTAL	0.208	0.208	0.171	0.221	0.223	**0.225**
EAR	NASA	0.268	0.262	0.235	0.244	0.230	**0.305**
	AEEEM	0.303	0.322	0.325	0.333	0.274	**0.404**
	TOTAL	0.286	0.292	0.280	0.289	0.252	**0.355**
EAF	NASA	0.218	0.218	0.178	0.218	0.169	**0.245**
	AEEEM	0.273	0.287	0.273	0.292	0.242	**0.359**
	TOTAL	0.246	0.253	0.226	0.255	0.206	**0.302**

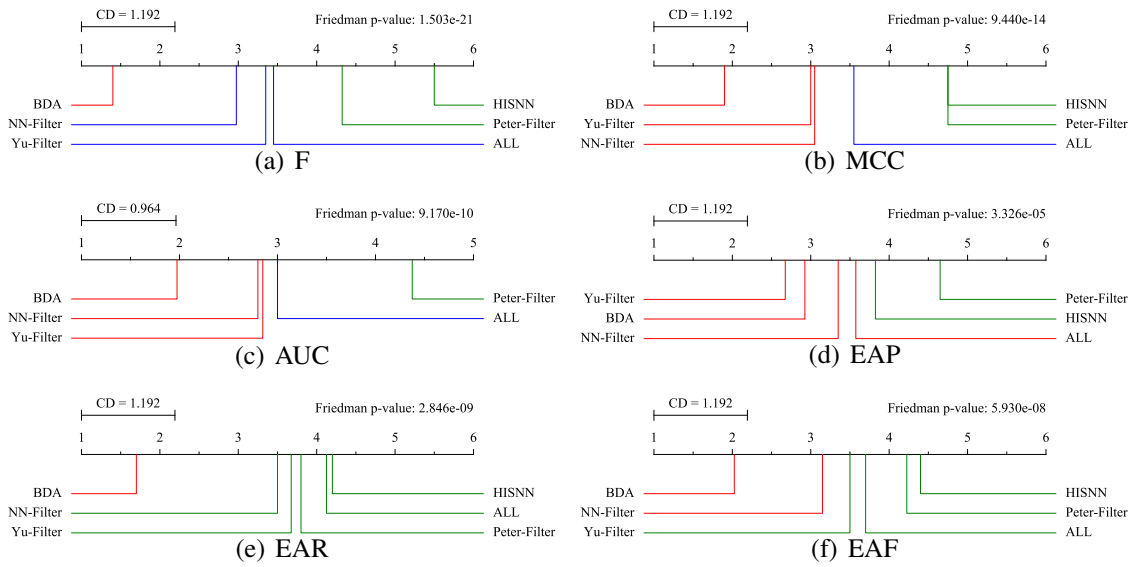


Figure 5.3: Comparison of BDA and five training data filter methods with Friedman test and Nemenyi post-hoc test in terms of all six indicators.

Results: Table 5.3 presents the average indicator values of BDA method and five training data filter methods on the cross project pairs of NASA dataset, AEEEM dataset, and across the two datasets. Figure 5.3 depicts the statistic test results on the 40 cross project pairs across the two datasets. From Table 5.3 and Figure 5.3, we have the following observations.

First, from Table 5.3, on the 20 cross project pairs of NASA dataset, our BDA method achieves the best performance in terms of five indicators (except for EAP); on the 20 cross project pairs of AEEEM dataset, our BDA model also achieves the best performance in terms of five indicators (except for EAP); across all 40 cross project pairs, our BDA model achieves the best performance in terms of all indicators.

Second, compared with the five baseline methods, for the average indicator values of BDA across the two datasets, BDA achieves average improvements of 46.2%, 87.1%, 13.1%, 10.4%, 27.3%, and 28.4% in terms of F, MCC, AUC, EAP, EAR, and EAF, respectively. Compared with the best average indicator values among the five baseline

methods, BDA achieves average improvements of 22.2%, 35.9%, 6.9%, 1.1%, 21.6%, and 18.4% in terms of F, MCC, AUC, EAP, EAR, and EAF, respectively.

Third, from Figure 5.3, we observe that the p values of Friedman test in all subfigures are all less than 0.05, which means that there exist significant performance differences among the six methods. Our BDA model always belongs to the top-ranked group in terms of all indicators and ranks the first in terms of five indicators (except for EAP). In addition, BDA is significantly superior to all five baseline methods in terms of two indicators, i.e., F and EAR. But BDA has no significant differences compared with 2, 2, 3, and 1 baseline methods in terms of MCC, AUC, EAP, and EAF, respectively.

Discussion: The reason why BDA is superior to the training data filter methods is that BDA transforms the source project data and target project data into a common feature space to maximize their distributional similarity, while the training data filter methods only select a module subset from the source project without conduct feature transformation which cannot guarantee the similarity of data distribution between the source and target projects.

Answer to RQ2: In summary, the cross project data whose features are mapped by the BDA model can more promote the CPDP performance than the cross project data obtained by the training data filter methods.

5.4.3 Results for RQ3

Methods: To answer this question, we select six transfer learning based baseline methods for comparison. The brief descriptions of these baseline methods are as follows: **T**ransfer **C**omponent **A**nalysis (**TCA**) method [104] only considers the margin distribution differences across the project data. Before performing TCA, TCA+ method [102] selects

a specific data normalization strategy based on some designed rules to preprocess the data of the two projects. We design **C**onditional **D**istribution based **T**ransfer learning (**CDT**) method that only considers the conditional distribution differences between the data of the two projects for comparison. JDT (called JDA in [81]) is a joint distributions based transfer learning method that considers both the marginal and conditional distribution differences with equal weights. **T**ransfer **N**aive **B**ayes (**TNB**) [86] introduces the weight information of the modules into the Bayes formula. **Fe**ature **S**election using **C**lusters of **H**ybrid data (**FeSCH**) method [103] is a two-step feature selection based transfer learning method. This method consists of a feature clustering stage with a density-based clustering method and a feature selection stage with a ranking strategy.

Table 5.4: Average Indicator Values of BDA Model and Six Transfer Learning Methods on Each Dataset and Across All Datasets

Indicator	Dataset	TCA	TCA+	CDT	JDT	TNB	FeSCH	BDA
F	NASA	0.454	0.338	0.466	0.459	0.398	0.461	**0.489**
	AEEEM	0.512	0.470	0.514	0.519	0.536	0.475	**0.541**
	TOTAL	0.483	0.404	0.490	0.489	0.467	0.468	**0.515**
MCC	NASA	0.215	0.143	0.224	0.220	0.196	0.225	**0.242**
	AEEEM	0.303	0.277	0.298	0.299	0.234	0.270	**0.317**
	TOTAL	0.259	0.210	0.261	0.260	0.215	0.248	**0.280**
AUC	NASA	0.715	0.644	0.717	0.720	0.708	**0.726**	0.722
	AEEEM	**0.746**	0.728	0.735	0.741	0.725	0.703	**0.746**
	TOTAL	0.731	0.686	0.726	0.731	0.717	0.715	**0.734**
EAP	NASA	0.135	0.105	0.140	0.136	**0.164**	0.161	0.140
	AEEEM	0.305	0.292	0.300	0.301	0.207	0.297	**0.310**
	TOTAL	0.220	0.199	0.220	0.219	0.186	**0.229**	0.225
EAR	NASA	0.242	0.171	0.267	0.249	0.248	**0.330**	0.305
	AEEEM	0.363	0.357	0.375	0.380	0.374	0.344	**0.404**
	TOTAL	0.303	0.264	0.321	0.315	0.311	0.337	**0.355**
EAF	NASA	0.206	0.148	0.223	0.210	0.222	**0.268**	0.245
	AEEEM	0.326	0.320	0.337	0.341	0.297	0.308	**0.359**
	TOTAL	0.266	0.234	0.280	0.276	0.260	0.288	**0.302**

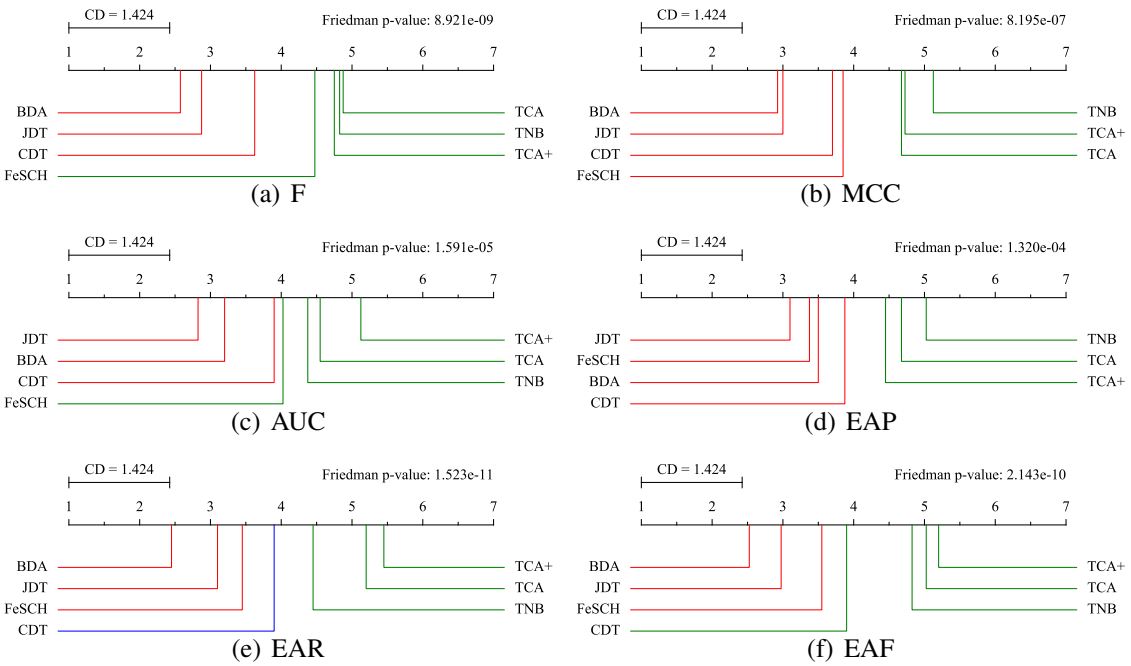


Figure 5.4: Comparison of BDA and six transfer learning methods with Friedman test and Nemenyi post-hoc test in terms of all six indicators.

Results: Table 5.4 presents the average indicator values of BDA method and six transfer learning methods on the cross project pairs of NASA dataset, AEEEM dataset, and across the two datasets. Figure 5.4 depicts the statistic test results on the 40 cross project pairs across the two datasets. From Table 5.4 and Figure 5.4, we have the following observations.

First, from Table 5.4, on the 20 cross project pairs of NASA dataset, our BDA method achieves the best performance in terms of two indicators (i.e., F and MCC); on the 20 cross project pairs of AEEEM dataset, our BDA model also achieves the best performance in terms of all six indicators; across all 40 cross project pairs, our BDA model achieves the best performance in terms of five indicators (except for EAP).

Second, compared with the five baseline methods, for the average indicator values of BDA across the two datasets, BDA achieves average improvements of 10.8%, 16.7%,

2.4%, 15.8%, and 13.6% in terms of F, MCC, AUC, EAR, and EAF, respectively. Compared with the best average indicator values among the five baseline methods, BDA achieves average improvements of 5.1%, 7.3%, 0.5%, 5.3%, and 4.9% in terms of F, MCC, AUC, EAR, and EAF, respectively. In addition, the baseline method FeSCH achieves the best average EAP value which is 1.7% better than our BDA model.

Third, from Figure 5.4, we observe that the p values of Friedman test in all subfigures are all less than 0.05, which means that there exist significant performance differences among the six methods. Our BDA model always belongs to the top-ranked group in terms of all indicators. In addition, BDA is significantly superior to all five baseline methods in terms of two indicators, i.e., F and EAR. But BDA has no significant differences compared with 2, 3, 2, 3, 2, and 2 baseline methods in terms of F, MCC, AUC, EAP, EAR, and EAF, respectively.

Discussion: The fact that BDA is superior to TCA and CDT indicates that the transfer learning method that considers two distribution differences and their weights can more promote the CPDP performance than the methods that only consider one distribution difference. The fact that BDA performs better than JDT indicates that considering the weights of the two distribution differences can further improve the CPDP performance.

Answer to RQ3: To sum up, compared with the methods that consider only one distribution difference and the method that uses the same weight to combine the two distribution differences, the method that consider both two distribution difference and their weights is more appropriate for the CPDP task.

5.4.4 Results for RQ4

Methods: To answer this question, we set 20 different thresholds for feature dimensions, from 5% to 100% with a step of 5%, for comparison.

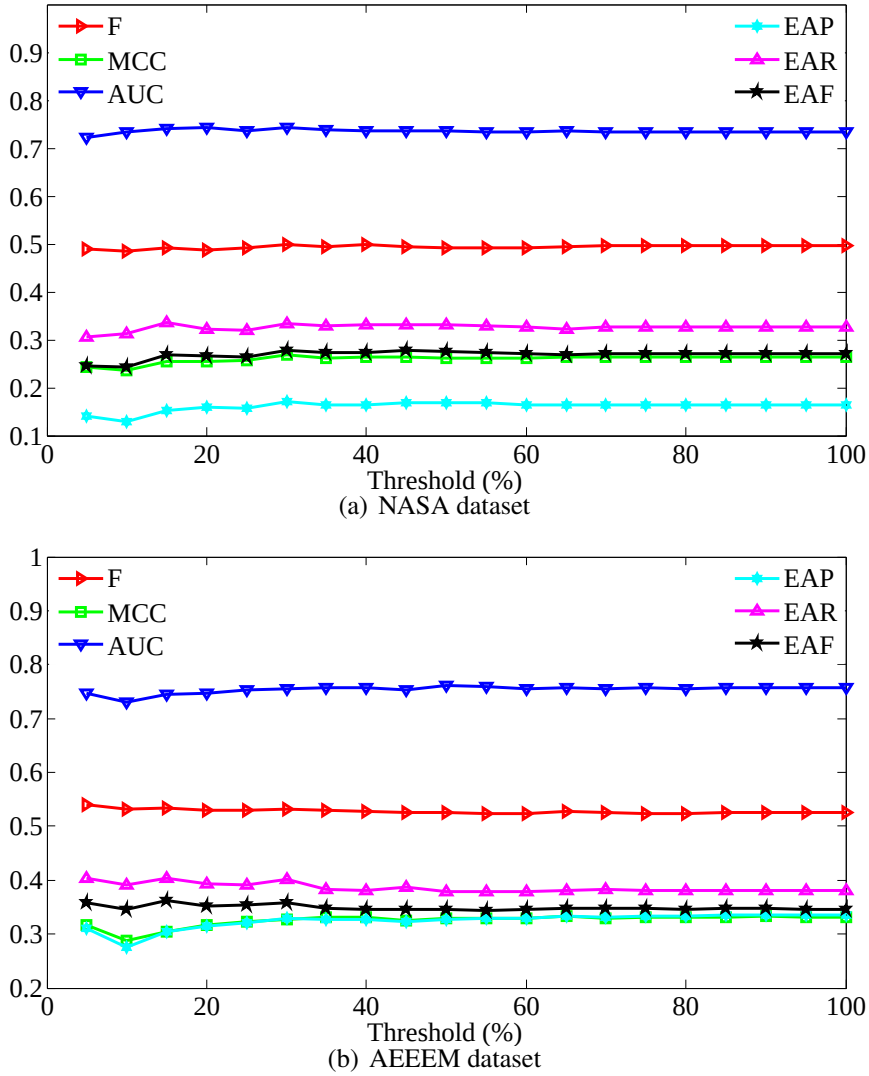


Figure 5.5: The six average indicator values of BDA when vary the thresholds of feature dimension on each benchmark dataset.

Results: Figure 5.5 depicts the six average indicator values of BDA when varying the thresholds of feature dimensions on each benchmark dataset.

From this figure, we observe that on NASA dataset, the six average indicator values of the BDA model increase with the increasing of feature dimensions under thresholds smaller than 20%, but the values keep almost unchanged under thresholds larger than 20%. On AEEEM dataset, the six average indicator values keep stable under thresholds larger than 15%.

Overall, the feature dimension has little impact on the performance of the BDA model under thresholds larger than 20%. When the threshold smaller than 20%, the BDA model can achieve comparable performance when the threshold is set to 5% compared with the performance under larger thresholds. From the above analysis, the thresholds with 5% can be a good choice for the BDA model to achieve the acceptable CPDP performance.

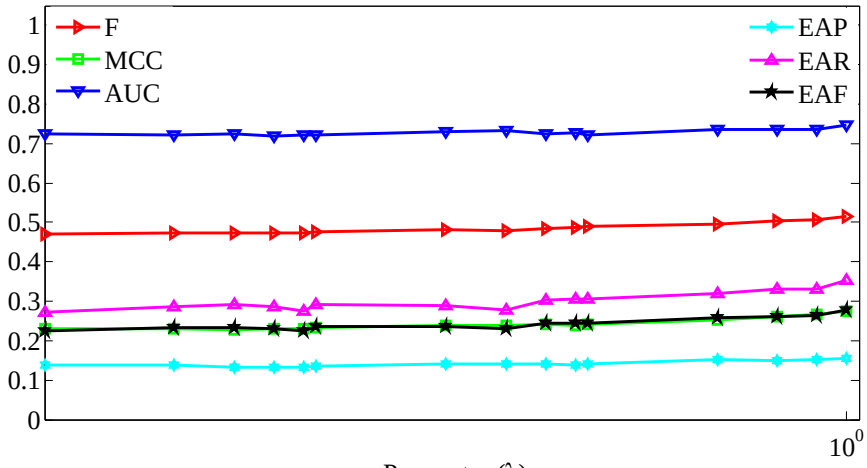
5.4.5 Results for RQ5

Methods: To answer this question, we empirically set 15 different parameter λ values, including 0.001$\sim$0.009 with a step of 0.002, 0.01$\sim$0.09 with a step of 0.02, and 0.1$\sim$0.9 with a step of 0.2 for comparison.

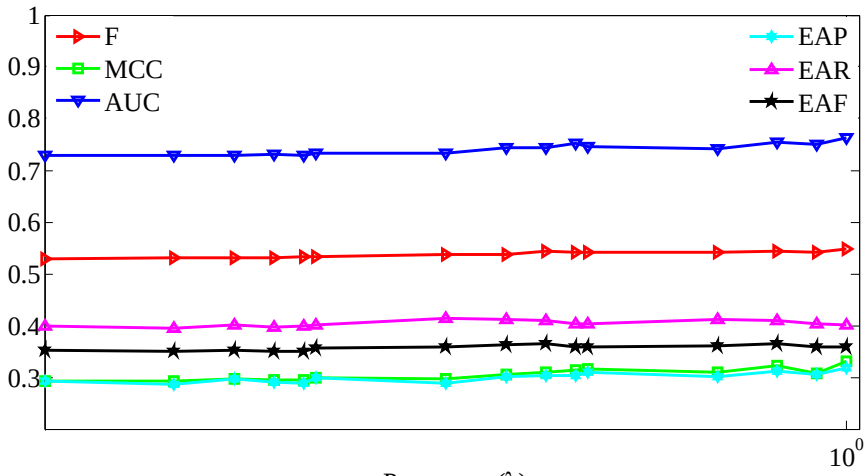
Results: Figure 5.6 depicts the six average indicator values of BDA with different λ values on each benchmark dataset.

From this figure, we observe that on NASA dataset, the six average indicator values of the BDA model increase with the increase of parameter λ values. On AEEEM dataset, the average values of MCC, AUC, and EAP increase with the increasing of parameter λ values while the average values of other three indicators are not impacted by the parameter λ values.

Overall, selecting the larger parameter λ value is helpful to improve the CPDP performance of our BDA model.



(a) NASA dataset



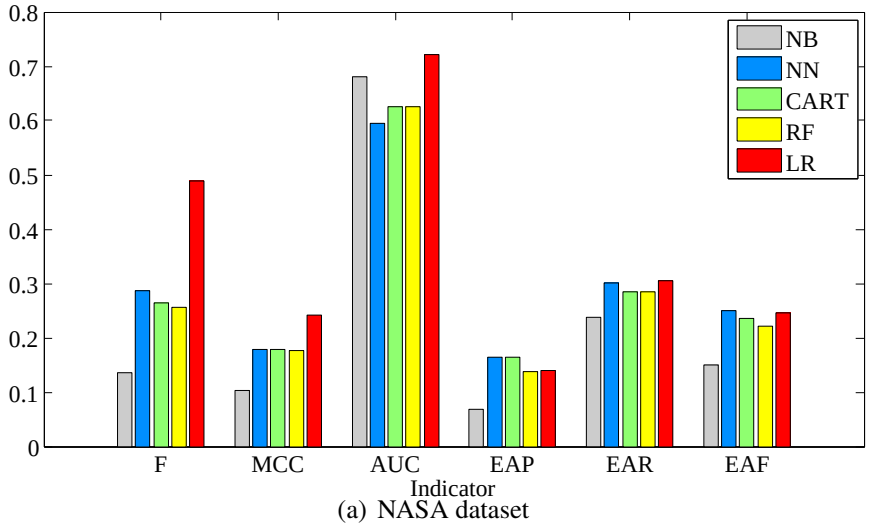
(b) AEEEM dataset

Figure 5.6: The six average indicator values of BDA when vary the thresholds of parameter λ on each benchmark dataset.

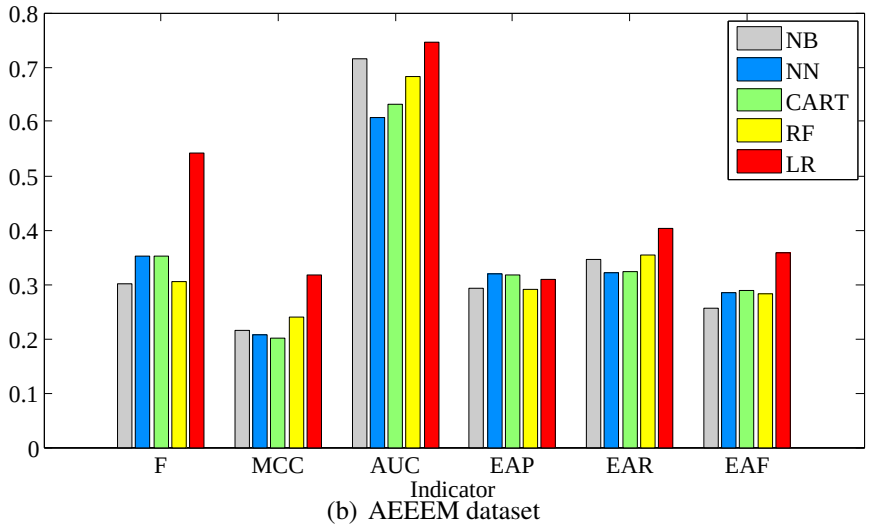
5.4.6 Results for RQ6

Methods: To answer this question, we choose four classic classifiers, including NB, NN, RF, and CART for observations.

Results: Figure 5.7 depicts the six average indicator values of BDA with different classifiers on each benchmark dataset.



(a) NASA dataset



(b) AEEEM dataset

Figure 5.7: The six average indicator values of BDA with different classifiers on each benchmark dataset.

From this figure, we observe that on NASA dataset, the three average traditional indicator values of the BDA model with LR classifier is obviously superior to that of the BDA model with other classifiers. In addition, the average EAR and EAF values of the BDA model with LR classifier is similar to that of the BDA model with NN classifier, but the average EAP vlaue of BDA model with LR classifier is lower than that of the BDA model with NN and CART classifiers. On AEEEM dataset, the average values of five indicators (except for EAP) of the BDA model with LR classifier is obviously superior to

that of the BDA model with other classifiers, but the average EAP value of the BDA model with LR classifier is similar to that of the BDA model with NN and CART classifiers.

Overall, in order to obtain better CPDP performance, combining our BDA model with LR classifier is the optimal choice.

5.5 Conclusion

In this chapter, we propose a transfer learning based CPDP framework by introducing an advanced balanced distribution adaption BDA model. The BDA model not only considers the marginal distribution difference of data between different projects, and also the conditional distribution difference. In addition, BDA model does not simply combine the two distribution differences, but assigns the two differences with distinct weights for different cross project pairs, aiming to adapting the degree of similarity of different cross project data. We compare our BDA model with five training data filter methods and six transfer learning methods on five projects of NASA dataset and five projects of AEEEM dataset. We use six indicators to evaluate the performance of these methods. The experimental results show that our proposed BDA model based CPDP framework can achieve better performance in most cases.

Chapter 6

Conclusion

6.1 Conclusion

Software has become ubiquitous in aspects of the society, politics, economy and military, and our daily life also relies increasingly on the software. Developing high quality software has always been the goal pursued by software developers. However, due to some uncontrollable factors in the process of software design, development and configuration, software inevitably contains defects which can cause bad consequences with different degrees. Thus, it is critical for the improvement of the software reliability by detecting as many defective software modules as possible and fixing them before delivering the software product. For this purpose, the researchers proposed different defect prediction techniques. They uses different statistic learning or machine learning methods to identify the software modules that are most likely to contain defects by mining the historical development data in the software repository. These high-risk software modules can be recommended to software developers or testers for priority review, which greatly optimizes the configuration of software testing resources, improves the efficiency of software development and promotes the software quality assurance.

In this thesis, we mainly carry out in-depth researches on the difficulties faced by three defect prediction scenarios (including IVDP, CVDP, and CPDP), and apply different machine learning methods, including kernel learning, class imbalanced learning, sparse representation learning and transfer learning methods, to assist in solving the corresponding difficulties. The main work includes the following three aspects:

For the feature extraction and class imbalance issue under the IVDP scenario, we propose a KPWE framework that combines the feature learning based on kernel principal component analysis (KPCA) and class imbalanced learning based on weighted extreme learning machine (WELM). In order to make it easier to distinguish software modules with different labels in defect data, this framework first uses KPCA to map the original features into a high-dimensional space, making the software modules linearly separable. Then, it employs WELM to construct the classification model on the mapped training set and the model is used to conduct prediction task on the mapped test set. WELM alleviates the negative effects of class imbalance by assigning different weights to the software modules with distinct labels. The experimental results on 10 projects from NASA benchmark dataset and five projects from AEEEM benchmark dataset show that our proposed IVDP framework is generally superior to 24 baseline methods.

For the training subset selection issue under the CVDP scenario, we propose an optimization based two-stage training subset selection framework, called TSTSS. In the first stage, TSTSS uses a sparse modeling representation selection (SMRS) method to select a simplified subset to represent the original defect data of the previous version without the participation of the defect data of the current version. The simplified subset can minimize the error for reconstructing the original defect data. In the second stage, TSTSS applies a dissimilarity-based spare subset selection sparse (DS3) method to select a refined subset from the simplified subset in the previous step with the assistance of the defect data of the current version. The obtained final module subset can well represent the

defect data of the current version. The experimental results on 50 cross-version pairs from 67 versions of 17 projects in the PROMISE benchmark dataset illustrate that our CVDP framework performs better than 11 baseline methods on the whole.

For the data distribution difference issue under CPDP scenario, we propose a transfer learning based framework by introducing a novel balanced distribution adaptation BDA model. This model combines both marginal and conditional distribution differences across project data, and also considers the importance degrees of the two differences over different cross-project pairs. The experimental results on 40 cross project pairs from five projects in the NASA benchmark dataset and five projects in the AEEEM benchmark dataset manifest that our proposed CPDP framework achieves better performance than 11 baseline methods overall.

6.2 Future Work

In this thesis, we analyze and study some issues under 3 different defect prediction scenarios, and propose feasible solutions to solve these problems. Although some progress has been made in the current work, some aspects still need to be expanded and improved. In addition, some work has not been studied yet and further exploration is needed. Here, we list some work to be done in the future as follows:

In chapter 5, we mainly focuses on using machine learning methods to reduce the distribution differences among cross-project data and investigating the performance of the transfer learning method based cross-project prediction model. But we do not take the impact of class imbalance on the prediction performance into account as in Chapter 3 and Chapter 4. Therefore, in the future work, we intend to address the class imbalance issue during the process of knowledge transfer between the source and target project data.

In Chapter 5, the considered CPDP scenario assumes that the feature sets of the two projects are homogeneous. But in practice, due to the different project development platforms, the varying development languages, and the distinct feature extraction tools, the final extracted feature sets of distinct projects are usually different. In this case, besides the data distributions across projects are different, the feature heterogeneous issue also need to be solved. Thus, the heterogeneous CPDP issue is more difficult than the homogeneous CPDP. Considering that there are not many researches on this topic at present, in the future work, we will explore new machine learning methods to solve the difficulties under heterogeneous CPDP scenarios.

Since the defect prediction tasks under the three scenarios in this thesis all require the labeled training set to construct the classification model, they belong to the category of supervised defect prediction. Compared with the supervised model, the unsupervised model does not require the participation of the labeled software modules, which is helpful to reduce the labeling cost of constructing the labeled training data. However, the unsupervised defect prediction researches have not received as much attention as the supervised defect prediction researches. Therefore, in future, we will carry out relevant work on unsupervised defect prediction.

Currently, Android applications are becoming more and more popular in our daily life, and its reliability has been widely concerned. However, the defect data used in this thesis are all from traditional software projects. Therefore, in the future work, we plan to do some relevant analysis and researches on defect prediction problems for Android software projects.

The studies in this thesis mainly apply different machine learning methods to determine whether a software module contains defects, regardless of the type of defect belongs to. In the future research, we will take this point into consideration, i.e., carrying out relevant

researches on software defect type prediction.

This thesis mainly treats defect prediction task as a machine learning problem and does not consider the interaction between software defect prediction and software design, development, testing and maintenance processes. In future studies, we plan to combine the results of defect prediction with software development process for discussion, aiming to give full play to the application value of machine learning assisted defect prediction methods.

Bibliography

[1] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[2] Supreeth Achar, Bharath Sankaran, Stephen Nuske, Sebastian Scherer, and Sanjiv Singh. Self-supervised segmentation of river scenes. In *2011 IEEE International Conference on Robotics and Automation*, pages 6227–6232. IEEE, 2011.

[3] Josephine Akosa. Predictive accuracy: a misleading performance measure for highly imbalanced data. In *Proceedings of the SAS Global Forum*, pages 2–5, 2017.

[4] Erik Arisholm, Lionel C Briand, and Eivind B Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software (JSS)*, 83(1):2–17, 2010.

[5] Arthur Asuncion and David Newman. Uci machine learning repository, 2007.

[6] Kwabena Ebo Bennin, Jacky Keung, and Akito Monden. Impact of the distribution parameter of data sampling approaches on software defect prediction models. In *Proceedings of the 24th Asia-Pacific Software Engineering Conference (APSEC)*, pages 630–635. IEEE, 2017.

[7] Kwabena Ebo Bennin, Jacky Keung, Akito Monden, Passakorn Phannachitta, and Solomon Mensah. The significant effects of data sampling approaches on software defect prioritization and classification. In *Proceedings of the 11th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 364–373. IEEE Press, 2017.

[8] Kwabena Ebo Bennin, Jacky Keung, Passakorn Phannachitta, Akito Monden, and Solomon Mensah. Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Transactions on Software Engineering (TSE)*, 44(6):534–550, 2017.

[9] Kwabena Ebo Bennin, Koji Toda, Yasutaka Kamei, Jacky Keung, Akito Monden, and Naoyasu Ubayashi. Empirical evaluation of cross-release effort-aware defect prediction models. In *Proceedings of 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 214–221. IEEE, 2016.

[10] Yi Bin, Kai Zhou, Hongmin Lu, Yuming Zhou, and Baowen Xu. Training data selection for cross-project defection prediction: which approach is better? In *Proceedings of the 11th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 354–363, 2017.

[11] Partha S Bishnu and Vandana Bhattacherjee. Software fault prediction using quad tree-based k-means clustering algorithm. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(6):1146–1150, 2011.

[12] Remco R Bouckaert, Eibe Frank, Mark Hall, Richard Kirkby, Peter Reutemann, Alex Seewald, and David Scuse. Weka manual for version 3-6-1. *The University of Waikato: Hamilton, New Zealand*, pages 1–341, 2009.

[13] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[14] Lionel C Briand, Walcelio L. Melo, and Jurgen Wust. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Transactions on Software Engineering (TSE)*, 28(7):706–720, 2002.

[15] Hui Cao, Zheng Qin, and Tao Feng. A novel pca-bp fuzzy neural network model for software defect prediction. *Advanced Science Letters*, 9(1):423–428, 2012.

[16] Cagatay Catal and Banu Diri. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences*, 179(8):1040–1058, 2009.

[17] Cagatay Catal and Banu Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346–7354, 2009.

[18] Gemma Catolino, Fabio Palomba, Andrea De Lucia, Filomena Ferrucci, and Andy Zaidman. Developer-related factors in change prediction: an empirical assessment. In *Proceedings of the 25th International Conference on Program Comprehension (ICPC)*, pages 186–195, 2017.

[19] Lin Chen, Bin Fang, Zhaowei Shang, and Yuanyan Tang. Negative samples reduction in cross-company software defects prediction. *Information and Software Technology (IST)*, 62:67–77, 2015.

[20] Mingming Chen and Yutao Ma. An empirical study on predicting defect numbers. In *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 397–402, 2015.

[21] Xiang Chen, Yuxiang Shen, Zhanqi Cui, and Xiaolin Ju. Applying feature selection to software defect prediction using multi-objective optimization. In *Proceedings of the 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 54–59. IEEE, 2017.

[22] Rodrigo A Coelho, Fabrício dos RN Guimarães, and Ahmed AA Esmin. Applying swarm ensemble clustering technique for fault prediction using software metrics. In *Proceedings of the 13th International Conference on Machine Learning and Applications*, pages 356–361. IEEE, 2014.

[23] Marco DAmbros, Michele Lanza, and Romain Robbes. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering (EMSE)*, 17(4-5):531–577, 2012.

[24] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(Jan):1–30, 2006.

[25] N Dhamayanthi and B Lavanya. Improvement in software defect prediction outcome using principal component analysis and ensemble machine learning algorithms. In *International Conference on Intelligent Data Communication Technologies and Internet of Things*, pages 397–406. Springer, 2018.

[26] Shifei Ding, Han Zhao, Yanan Zhang, Xinzheng Xu, and Ru Nie. Extreme learning machine: algorithm, theory and applications. *Artificial Intelligence Review*, 44(1):103–115, 2015.

[27] Ehsan Elhamifar, Guillermo Sapiro, and S Shankar Sastry. Dissimilarity-based sparse subset selection. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 38(11):2182–2197, 2016.

[28] Ehsan Elhamifar and René Vidal. Sparse subspace clustering. In *Proceedings of the 22nd International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2790–2797, 2009.

[29] Emelie Engström and Per Runeson. Software product line testing–a systematic mapping study. *Information and Software Technology (IST)*, 53(1):2–13, 2011.

[30] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[31] Wei Fu and Tim Menzies. Revisiting unsupervised learning for defect prediction. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (FSE)*, pages 72–83, 2017.

[32] Daniel Gabay and Bertrand Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17–40, 1976.

[33] Kehan Gao, Taghi M Khoshgoftaar, Huanjing Wang, and Naeem Seliya. Choosing software metrics for defect prediction: an investigation on feature selection techniques. *Software: Practice and Experience (SPE)*, 41(5):579–606, 2011.

[34] Baljinder Ghotra, Shane McIntosh, and Ahmed E Hassan. Revisiting the impact of classification techniques on the performance of defect prediction models. In *Proceedings of the 37th IEEE International Conference on Software Engineering (ICSE)*, volume 1, pages 789–800. IEEE, 2015.

[35] Baljinder Ghotra, Shane McIntosh, and Ahmed E Hassan. A large-scale study of the impact of feature selection techniques on defect classification models. In *Proceedings of the 14th International Conference on Mining Software Repositories (MSR)*, pages 146–157. IEEE, 2017.

[36] David Gray, David Bowes, Neil Davey, Yi Sun, and Bruce Christianson. The misuse of the nasa metrics data program data sets for automated software defect prediction. In *Proceedings of the 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE)*, pages 96–103. IET, 2011.

[37] Lan Guo, Yan Ma, Bojan Cukic, and Harshinder Singh. Robust prediction of fault-proneness by random forests. In *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE)*, pages 417–428. IEEE, 2004.

[38] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering (TSE)*, 38(6):1276–1304, 2011.

[39] Peng He, Yao He, Lvjun Yu, and Bing Li. An improved method for cross-project defect prediction by simplifying training data. *Mathematical Problems in Engineering*, 2018, 2018.

[40] Zhimin He, Fengdi Shu, Ye Yang, Mingshu Li, and Qing Wang. An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering (ASE)*, 19(2):167–199, 2012.

[41] Steffen Herbold. Training data selection for cross-project defect prediction. In *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*, page 6, 2013.

[42] Steffen Herbold, Alexander Trautsch, and Jens Grabowski. A comparative study to benchmark cross-project defect prediction approaches. *IEEE Transactions on Software Engineering (TSE)*, 44(9):811–833, 2017.

[43] Steffen Herbold, Alexander Trautsch, and Jens Grabowski. A comparative study to benchmark cross-project defect prediction approaches. *IEEE Transactions on Software Engineering (TSE)*, 44(9):811–833, 2017.

[44] Kim Herzig, Sascha Just, Andreas Rau, and Andreas Zeller. Predicting defects using change genealogies. In *Proceedings of the 24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 118–127. IEEE, 2013.

[45] Tilman Holschuh, Markus Pauser, Kim Herzig, Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. Predicting defects in sap java code: An experience report. In *Proceedings of the 31st International Conference on Software Engineering (ICSE)*, pages 172–181, 2009.

[46] Seyedrebvar Hosseini, Burak Turhan, and Dimuthu Gunarathna. A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Transactions on Software Engineering (TSE)*, 45(2):111–147, 2017.

[47] Jaroslaw Hryszko, Lech Madeyski, Marta Dabrowska, and Piotr Konopka. Defect prediction with bad smells in code. *arXiv preprint arXiv:1703.06300*, 2017.

[48] Gao Huang, Guangbin Huang, Shiji Song, and Keyou You. Trends in extreme learning machines: A review. *Neural Networks*, 61:32–48, 2015.

[49] Guangbin Huang, Lei Chen, and Chee Kheong Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks (TNN)*, 17(4):879–892, 2006.

[50] Guangbin Huang, Qinyu Zhu, and Cheekheong Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489–501, 2006.

[51] Qiao Huang, Xin Xia, and David Lo. Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction. In *Proceedings of the 33rd International Conference on Software Maintenance and Evolution (ICSME)*, pages 159–170, 2017.

[52] Qiao Huang, Xin Xia, and David Lo. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. *Empirical Software Engineering*, 24(5):2823–2862, 2019.

[53] Sheng-Jun Huang, Rong Jin, and Zhi-Hua Zhou. Active learning by querying informative and representative examples. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 10(36):1936–1949, 2014.

[54] Tian Jiang, Lin Tan, and Sunghun Kim. Personalized defect prediction. In *Proceedings of the 28th International Conference on Automated Software Engineering (ASE)*, pages 279–289, 2013.

[55] Yue Jiang, Bojan Cukic, and Yan Ma. Techniques for evaluating fault prediction models. *Empirical Software Engineering (EMSE)*, 13(5):561–595, 2008.

[56] Xiao-Yuan Jing, Fei Wu, Xiwei Dong, and Baowen Xu. An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Transactions on Software Engineering (TSE)*, 43(4):321–339, 2017.

[57] Xiao-Yuan Jing, Shi Ying, Zhi-Wu Zhang, Shan-Shan Wu, and Jin Liu. Dictionary learning based software defect prediction. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pages 414–423. ACM, 2014.

[58] Xiaoyuan Jing, Fei Wu, Xiwei Dong, Fumin Qi, and Baowen Xu. Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (FSE)*, pages 496–507, 2015.

[59] Charles R Johnson. *Matrix theory and applications*, volume 40. American Mathematical Soc., 1990.

[60] Yasutaka Kamei, Akito Monden, Shinsuke Matsumoto, Takeshi Kakimoto, and Ken-ichi Matsumoto. The effects of over and under sampling on fault-prone module detection. In *Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 196–204. IEEE, 2007.

[61] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering (TSE)*, 39(6):757–773, 2013.

[62] Kazuya Kawata, Sousuke Amasaki, and Tomoyuki Yokogawa. Improving relevancy filter methods for cross-project defect prediction. In *Applied Computing & Information Technology*, pages 1–12. 2016.

[63] Taghi M Khoshgoftaar, Erik Geleyn, Laurent Nguyen, and Lofton Bullard. Cost-sensitive boosting in software quality modeling. In *Proceedings of the 7th International Symposium on High Assurance Systems Engineering*, pages 51–60. IEEE, 2002.

[64] Taghi M Khoshgoftaar and Naeem Seliya. Fault prediction modeling for software quality estimation: Comparing commonly used techniques. *Empirical Software Engineering (EMSE)*, 8(3):255–283, 2003.

[65] Taghi M Khoshgoftaar, Ruqun Shan, and Edward B Allen. Improving tree-based models of software quality with principal components analysis. In *Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE)*, pages 198–209. IEEE, 2000.

[66] Kwang In Kim, Matthias O Franz, and Bernhard Scholkopf. Iterative kernel principal component analysis for image modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 27(9):1351–1366, 2005.

[67] Issam H Laradji, Mohammad Alshayeb, and Lahouari Ghouti. Software defect prediction using ensemble learning on selected features. *Information and Software Technology (IST)*, 58:388–402, 2015.

[68] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. Benchmarking classification models for software defect prediction: A proposed

framework and novel findings. *IEEE Transactions on Software Engineering (TSE)*, 34(4):485–496, 2008.

[69] Hui Li, Guofeng Gao, Rong Chen, Xin Ge, Shikai Guo, and Li-Ying Hao. The influence ranking for testers in bug tracking systems. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 29(01):93–113, 2019.

[70] JunBao Li, Shuchuan Chu, and Jeng-Shyang Pan. Kernel principal component analysis (kpca)-based face recognition. In *Kernel Learning Algorithms for Face Recognition*, pages 71–99. Springer, 2014.

[71] Ming Li, Hongyu Zhang, Rongxin Wu, and Zhi-Hua Zhou. Sample-based software defect prediction with active and semi-supervised learning. *Automated Software Engineering (ASE)*, 19(2):201–230, 2012.

[72] Xin Li and Yuhong Guo. Adaptive active learning for image classification. In *Proceedings of the 26th Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 859–866, 2013.

[73] Yuting Li, Jianmin Su, and Xiaoxing Yang. Multi-objective vs. single-objective approaches for software defect prediction. In *Proceedings of the 2nd International Conference on Management Engineering, Software Engineering and Service Sciences*, pages 122–127, 2018.

[74] Zhiqiang Li, Xiao-Yuan Jing, Fei Wu, Xiaoke Zhu, Baowen Xu, and Shi Ying. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. *Automated Software Engineering (ASE)*, 25(2):201–245, 2018.

[75] Zhiqiang Li, Xiao-Yuan Jing, and Xiaoke Zhu. Heterogeneous fault prediction with cost-sensitive domain adaptation. *Software Testing, Verification and Reliability (STVR)*, 28(2):e1658, 2018.

[76] Zhiqiang Li, Xiao-Yuan Jing, Xiaoke Zhu, and Hongyu Zhang. Heterogeneous defect prediction through multiple kernel learning and ensemble learning. In *Proceedings of the 33rd International Conference on Software Maintenance and Evolution (ICSME)*, pages 91–102, 2017.

[77] Zhiqiang Li, Xiao-Yuan Jing, Xiaoke Zhu, Hongyu Zhang, Baowen Xu, and Shi Ying. On the multiple sources and privacy preservation issues for heterogeneous defect prediction. *IEEE Transactions on Software Engineering (TSE)*, 2017.

[78] Chao Liu, Dan Yang, Xin Xia, Meng Yan, and Xiaohong Zhang. A two-phase transfer learning model for cross-project defect prediction. *Information and Software Technology (IST)*, 107:125–136, 2019.

[79] Fang Liu, Xing Gao, Bing Zhou, and Juan Deng. Software defect prediction model based on pca-isvm. *Computer Simulation*, 2014.

[80] Mingxia Liu, Linsong Miao, and Daoqiang Zhang. Two-stage cost-sensitive learning for software defect prediction. *IEEE Transactions on Reliability (TR)*, 63(2):676–686, 2014.

[81] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. Transfer feature learning with joint distribution adaptation. In *Proceedings of the 14th International Conference on Computer Vision (ICCV)*, pages 2200–2207, 2013.

[82] Huihua Lu, Bojan Cukic, and Mark Culp. Software defect prediction using semi-supervised learning with dimension reduction. In *Proceedings of the 27th International Conference on Automated Software Engineering (ASE)*, pages 314–317. IEEE, 2012.

[83] Huihua Lu, Ekrem Kocaguneli, and Bojan Cukic. Defect prediction between software versions with active learning and dimensionality reduction. In *Proceedings of the 25th International Symposium on Software Reliability Engineering (ISSRE)*, pages 312–322, 2014.

[84] Guangchun Luo and Hao Chen. Kernel based asymmetric learning for software defect prediction. *IEICE Transactions on Information and Systems*, 95(1):267–270, 2012.

[85] Guangchun Luo, Ying Ma, and Ke Qin. Asymmetric learning based on kernel partial least squares for software defect prediction. *IEICE Transactions on Information and Systems*, 95(7):2006–2008, 2012.

[86] Ying Ma, Guangchun Luo, Xue Zeng, and Aiguo Chen. Transfer learning for cross-company software defect prediction. *Information and Software Technology (IST)*, 54(3):248–256, 2012.

[87] Lech Madeyski and Marian Jureczko. Which process metrics can significantly improve defect prediction models? an empirical study. *Software Quality Journal (SQJ)*, 23(3):393–422, 2015.

[88] Ruchika Malhotra. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27:504–518, 2015.

[89] Ruchika Malhotra. An empirical framework for defect prediction using machine learning techniques with android software. *Applied Soft Computing*, 49:1034–1050, 2016.

[90] Ruchika Malhotra and Rajeev Raje. An empirical comparison of machine learning techniques for software defect prediction. In *Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies*, pages 320–327. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[91] Thilo Mende and Rainer Koschke. Effort-aware defect prediction models. In *Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 107–116. IEEE, 2010.

[92] Tim Menzies, Kareem Ammar, Allen Nikora, and Justin DiStefano. How simple is software defect detection. *Submitted to the Emprical Software Engineering Journal*, 2003.

[93] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering (TSE)*, 33(1):2–13, 2007.

[94] Diego PP Mesquita, Lincoln S Rocha, João Paulo P Gomes, and Ajalmar R Rocha Neto. Classification with reject option for software defect prediction. *Applied Soft Computing*, 49:1085–1093, 2016.

[95] Akito Monden, Takuma Hayashi, Shoji Shinoda, Kumiko Shirai, Junichi Yoshida, Mike Barker, and Kenichi Matsumoto. Assessing the cost effectiveness of fault prediction in acceptance testing. *IEEE Transactions on Software Engineering (TSE)*, 39(10):1345–1357, 2013.

[96] John C. Munson and Taghi M. Khoshgoftaar. The detection of fault-prone programs. *IEEE Transactions on Software Engineering (TSE)*, 18(5):423, 1992.

[97] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[98] Nachiappan Nagappan and Thomas Ball. Static analysis tools as early indicators of pre-release defect density. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 580–586. IEEE, 2005.

[99] Nachiappan Nagappan and Thomas Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 284–292, 2005.

[100] Jaechang Nam, Wei Fu, Sunghun Kim, Tim Menzies, and Lin Tan. Heterogeneous defect prediction. *IEEE Transactions on Software Engineering (TSE)*, 44(9):874–896, 2017.

[101] Jaechang Nam and Sunghun Kim. Clami: Defect prediction on unlabeled datasets (t). In *Proceedings of the 30th International Conference on Automated Software Engineering (ASE)*, pages 452–463. IEEE, 2015.

[102] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. Transfer defect learning. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 382–391, 2013.

[103] Chao Ni, Wang-Shu Liu, Xiang Chen, Qing Gu, Dao-Xu Chen, and Qi-Guo Huang. A cluster based feature selection method for cross-project software defect prediction. *Journal of Computer Science and Technology (JCST)*, 32(6):1090–1107, 2017.

[104] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2010.

[105] Witold Pedrycz, Giancarlo Succi, Petr Musılek, and Xiao Bai. Using self-organizing maps to analyze object-oriented software measures. *Journal of Systems and Software (JSS)*, 59(1):65–82, 2001.

[106] Jing Peng and Douglas R Heisterkamp. Kernel indexing for relevance feedback image retrieval. In *Proceedings of the 10th International Conference on Image Processing (ICIP)*, volume 1, pages I–733. IEEE, 2003.

[107] Fayola Peters, Tim Menzies, and Andrian Marcus. Better cross company defect prediction. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, pages 409–418, 2013.

[108] Jean Petrić, David Bowes, Tracy Hall, Bruce Christianson, and Nathan 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)*, pages 1–10, 2016.

[109] Faimison Porto, Leandro Minku, Emilia Mendes, and Adenilso Simao. A systematic study of cross-project defect prediction with meta-learning. *arXiv preprint arXiv:1802.06025*, 2018.

[110] Haini Qu, Guozheng Li, and Weisheng Xu. An asymmetric classifier based on partial least squares. *Pattern Recognition*, 43(10):3448–3457, 2010.

[111] Foyzur Rahman, Daryl Posnett, and Premkumar Devanbu. Recalling the imprecision of cross-project defect prediction. In *Proceedings of the 20th International Symposium on the Foundations of Software Engineering (FSE)*, pages 1–11, 2012.

[112] Calyampudi Radhakrishna Rao and Sujit Kumar Mitra. Generalized inverse of matrices and its applications. 1971.

[113] Santosh S Rathore and Sandeep Kumar. An empirical study of some software fault prediction techniques for the number of faults prediction. *Soft Computing*, 21(24):7417–7434, 2017.

[114] Santosh Singh Rathore and Sandeep Kuamr. Comparative analysis of neural network and genetic programming for number of software faults prediction. In *Proceedings of the 2015 National Conference on Recent Advances in Electronics & Computer Engineering*, pages 328–332. IEEE, 2015.

[115] Santosh Singh Rathore and Sandeep Kumar. Predicting number of faults in software system using genetic programming. In *Proceedings of 2015 International Conference on Soft Computing and Software Engineering*, pages 303–311, 2015.

[116] Santosh Singh Rathore and Sandeep Kumar. A decision tree regression based approach for the number of software faults prediction. *ACM SIGSOFT Software Engineering Notes*, 41(1):1–6, 2016.

[117] Jinsheng Ren, Ke Qin, Ying Ma, and Guangchun Luo. On software defect prediction using machine learning. *Journal of Applied Mathematics*, 2014, 2014.

[118] Duksan Ryu, Okjoo Choi, and Jongmoon Baik. Value-cognitive boosting with a support vector machine for cross-project defect prediction. *Empirical Software Engineering (EMSE)*, 21(1):43–71, 2016.

[119] Duksan Ryu, Jong-In Jang, and Jongmoon Baik. A hybrid instance selection using nearest-neighbor for cross-project defect prediction. *Journal of Computer Science and Technology (JCST)*, 30(5):969–980, 2015.

[120] Duksan Ryu, Jong-In Jang, and Jongmoon Baik. A transfer cost-sensitive boosting approach for cross-project defect prediction. *Software Quality Journal (SQJ)*, 25(1):235–272, 2017.

[121] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *Proceedings of the 7th International Conference on Artificial Neural Networks (ICANN)*, pages 583–588. Springer, 1997.

[122] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[123] Naeem Seliya and Taghi M Khoshgoftaar. Software quality estimation with limited fault data: a semi-supervised learning perspective. *Software Quality Journal (SQJ)*, 15(3):327–344, 2007.

[124] Martin Shepperd, David Bowes, and Tracy Hall. Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering (TSE)*, 40(6):603–616, 2014.

[125] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. Data quality: Some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering (TSE)*, 39(9):1208–1215, 2013.

[126] Shivkumar Shivaji. *Efficient bug prediction and fix suggestions*. PhD thesis, University of California, Santa Cruz, 2013.

[127] Shivkumar Shivaji, E James Whitehead, Ram Akella, and Sunghun Kim. Reducing features to improve code change-based bug prediction. *IEEE Transactions on Software Engineering (TSE)*, 39(4):552–569, 2012.

[128] Shivkumar Shivaji, E James Whitehead, Ram Akella, and Sunghun Kim. Reducing features to improve code change-based bug prediction. *IEEE Transactions on Software Engineering (TSE)*, 39(4):552–569, 2013.

[129] Shivkumar Shivaji, Jr E James Whitehead, Ram Akella, and Sunghun Kim. Reducing features to improve bug prediction. In *Proceedings of the 24th International Conference on Automated Software Engineering (ASE)*, pages 600–604. IEEE Computer Society, 2009.

[130] Swapnil Shukla, T Radhakrishnan, K Muthukumaran, and Lalita Bhanu Murthy Neti. Multi-objective cross-version defect prediction. *Soft Computing*, 22(6):1959–1980, 2018.

[131] Michael J Siers and Md Zahidul Islam. Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem. *Information Systems*, 51:62–71, 2015.

[132] Qinbao Song, Zihan Jia, Martin Shepperd, Shi Ying, and Jin Liu. A general software defect-proneness prediction framework. *IEEE Transactions on Software Engineering (TSE)*, 37(3):356–370, 2010.

[133] Qinbao Song, Zihan Jia, Martin Shepperd, Shi Ying, and Jin Liu. A general software defect-proneness prediction framework. *IEEE Transactions on Software Engineering (TSE)*, 37(3):356–370, 2011.

[134] Qinbao Song, Jingjie Ni, and Guangtao Wang. A fast clustering-based feature subset selection algorithm for high-dimensional data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 25(1):1–14, 2013.

[135] Zhongbin Sun, Qinbao Song, and Xiaoyan Zhu. Using coding-based ensemble learning to improve software defect prediction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1806–1817, 2012.

[136] Chakkrit Tantithamthavorn, Ahmed E Hassan, and Kenichi Matsumoto. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Transactions on Software Engineering (TSE)*, 2018.

[137] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering (TSE)*, 43(1):1–18, 2016.

[138] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering (TSE)*, 43(1):1–18, 2017.

[139] Jeff Tian. *Software quality engineering: testing, quality assurance, and quantifiable improvement*. John Wiley & Sons, 2005.

[140] Haonan Tong, Bin Liu, and Shihai Wang. Kernel spectral embedding transfer ensemble for heterogeneous defect prediction. *IEEE Transactions on Software Engineering (TSE)*, 2019.

[141] Burak Turhan and Ayse Basar Bener. Software defect prediction: Heuristics for weighted naïve bayes. In *Proceedings of the Second International Conference on Software and Data Technologies*, pages 244–249, 2007.

[142] Burak Turhan, Tim Menzies, Ayşe B Bener, and Justin Di Stefano. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering (EMSE)*, 14(5):540–578, 2009.

[143] Giorgio Valentini and Francesco Masulli. Ensembles of learning machines. In *Italian workshop on neural nets*, pages 3–20. Springer, 2002.

[144] Rene Vidal, Guillermo Sapiro, and E Elhamifar. See all by looking at a few: Sparse modeling for finding representative objects. In *Proceedings of the 25th Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1600–1607, 2012.

[145] Jindong Wang, Yiqiang Chen, Shuji Hao, Wenjie Feng, and Zhiqi Shen. Balanced distribution adaptation for transfer learning. In *Proceedings of the 17th International Conference on Data Mining (ICDM)*, pages 1129–1134. IEEE, 2017.

[146] Shuo Wang and Xin Yao. Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability (TR)*, 62(2):434–443, 2013.

[147] Song Wang, Taiyue Liu, and Lin Tan. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 297–308, 2016.

[148] Tao Wang and Wei-hua Li. Naive bayes software defect prediction model. In *Proceedings of the 2010 International Conference on Computational Intelligence and Software Engineering*, pages 1–4. IEEE, 2010.

[149] Tiejian Wang, Zhiwu Zhang, Xiaoyuan Jing, and Liqiang Zhang. Multiple kernel ensemble learning for software defect prediction. *Automated Software Engineering (ASE)*, 23(4):569–590, 2016.

[150] Shinya Watanabe, Haruhiko Kaiya, and Kenji Kaijiri. Adapting a fault prediction model to allow inter language reuse. In *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, pages 19–24, 2008.

[151] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1-3):37–52, 1987.

[152] Xin Xia, David Lo, Sinno Jialin Pan, Nachiappan Nagappan, and Xinyu Wang. Hydra: Massively compositional model for cross-project defect prediction. *IEEE Transactions on Software Engineering (TSE)*, 42(10):977–998, 2016.

[153] Zhou Xu, Jin Liu, Xiapu Luo, and Tao Zhang. Cross-version defect prediction via hybrid active learning with kernel principal component analysis. In *Proceedings of the 25th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, page to appear, 2018.

[154] Zhou Xu, Jin Liu, Zijiang Yang, Gege An, and Xiangyang Jia. The impact of feature selection on defect prediction performance: An empirical comparison. In *Proceedings of the 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 309–320. IEEE, 2016.

[155] Zhou Xu, Jifeng Xuan, Jin Liu, and Xiaohui Cui. Michac: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering. In *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 370–381, 2016.

[156] Zhou Xu, Sizhe Ye, Tao Zhang, Zhen Xia, Shuai Pang, Yong Wang, and Yutian Tang. Mvse: Effort-aware heterogeneous defect prediction via multiple-view spectral embedding. In *Proceedings of 2019 International Conference on Software Quality, Reliability and Security (QRS)*, pages 10–17. IEEE, 2019.

[157] Zhou Xu, Tao Zhang, Jacky Keung, Meng Yan, Xiapu Luo, Xiaohong Zhang, Ling Xu, and Yutian Tang. Feature selection and embedding based cross project framework for identifying crashing fault residence. *Information and Software Technology*, page 106452, 2020.

[158] Jifeng Xuan, He Jiang, Yan Hu, Zhilei Ren, Weiqin Zou, Zhongxuan Luo, and Xindong Wu. Towards effective bug triage with software data reduction techniques. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 27(1):264–280, 2014.

[159] Meng Yan, Yicheng Fang, David Lo, Xin Xia, and Xiaohong Zhang. File-level defect prediction: Unsupervised vs. supervised models. In *Proceedings of the 11th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 344–353. IEEE, 2017.

[160] Jun Yang and Hongbing Qian. Defect prediction on unlabeled datasets by using unsupervised clustering. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 465–472. IEEE, 2016.

[161] Xiaoxing Yang, Ke Tang, and Xin Yao. A learning-to-rank approach to software defect prediction. *IEEE Transactions on Reliability (TR)*, 64(1):234–246, 2015.

[162] Xiaoxing Yang and Wushao Wen. Ridge and lasso regression models for cross-version defect prediction. *Transactions on Reliability (TR)*, 67(3):885–896, 2018.

[163] Xinli Yang, David Lo, Xin Xia, and Jianling Sun. Tlel: A two-layer ensemble learning approach for just-in-time defect prediction. *Information and Software Technology (IST)*, 87:206–220, 2017.

[164] Xinli Yang, David Lo, Xin Xia, Yun Zhang, and Jianling Sun. Deep learning for just-in-time defect prediction. In *Proceedings of the 2015 International Conference on Software Quality, Reliability and Security (QRS)*, pages 17–26. IEEE, 2015.

[165] Yibiao Yang, Mark Harman, Jens Krinke, Syed Islam, David Binkley, Yuming Zhou, and Baowen Xu. An empirical study on dependence clusters for effort-aware fault-proneness prediction. In *Proceedings of the 31st International Conference on Automated Software Engineering (ASE)*, pages 296–307, 2016.

[166] Yibiao Yang, Yuming Zhou, Jinping Liu, Yangyang Zhao, Hongmin Lu, Lei Xu, Baowen Xu, and Hareton Leung. Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 157–168, 2016.

[167] Yibiao Yang, Yuming Zhou, Hongmin Lu, Lin Chen, Zhenyu Chen, Baowen Xu, Hareton Leung, and Zhenyu Zhang. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? an empirical study. *IEEE Transactions on Software Engineering (TSE)*, 41(4):331–357, 2014.

[168] Yibiao Yang, Yuming Zhou, Hongmin Lu, Lin Chen, Zhenyu Chen, Baowen Xu, Hareton Leung, and Zhenyu Zhang. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? an empirical study. *IEEE Transactions on Software Engineering (TSE)*, 41(4):331–357, 2015.

[169] Wenchao Yu, Fuzhen Zhuang, Qing He, and Zhongzhi Shi. Learning deep representations via extreme learning machines. *Neurocomputing*, 149:308–315, 2015.

[170] Xiao Yu, Jin Liu, Zijiang Yang, Xiangyang Jia, Qi Ling, and Sizhe Ye. Learning from imbalanced data for predicting the number of software defects. In *Proceedings of the 28th International Symposium on Software Reliability Engineering (ISSRE)*, pages 78–89. IEEE, 2017.

[171] Xiao Yu, Peipei Zhou, Jiansheng Zhang, and Jin Liu. A data filtering method based on agglomerative clustering. In *Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 392–397, 2017.

[172] Jerrold H Zar et al. *Biostatistical analysis*. Pearson Education India, 1999.

[173] Feng Zhang, Quan Zheng, Ying Zou, and Ahmed E Hassan. Cross-project defect prediction using a connectivity-based unsupervised classifier. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 309–320. IEEE, 2016.

[174] Zhi-Wu Zhang, Xiao-Yuan Jing, and Tie-Jian Wang. Label propagation based semi-supervised learning for software defect prediction. *Automated Software Engineering (ASE)*, 24(1):47–69, 2017.

[175] Jun Zheng. Cost-sensitive boosting neural networks for software defect prediction. *Expert Systems with Applications*, 37(6):4537–4543, 2010.

[176] Cheng Zhong. Software quality prediction method with hybrid applying principal components analysis and wavelet neural network and genetic algorithm.

International Journal of Digital Content Technology and its Applications, 5(3), 2011.

[177] Shi Zhong, Taghi M Khoshgoftaar, and Naeem Seliya. Unsupervised learning for expert-based software quality estimation. In *Proceedings of the 8th IEEE International Symposium on High-Assurance Systems Engineering*, pages 149–155. Citeseer, 2004.

[178] Yuming Zhou, Yibiao Yang, Hongmin Lu, Lin Chen, Yanhui Li, Yangyang Zhao, Junyan Qian, and Baowen Xu. How far we have progressed in the journey? an examination of cross-project defect prediction. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27(1):1–51, 2018.

[179] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the SIGSOFT Symposium on The Foundations of Software Engineering (FSE)*, pages 91–100, 2009.

[180] Weiwei Zong, Guangbin Huang, and Yiqiang Chen. Weighted extreme learning machine for imbalance learning. *Neurocomputing*, 101:229–242, 2013.