Available online at www.ijpe-online.com

**International Journal of Performability Engineering**

vol. 14, no. 6, June 2018, pp. 1291-1299
DOI: 10.23940/ijpe.18.06.p20.12911299

# Impact of Hyper Parameter Optimization for Cross-Project Software Defect Prediction

Yubin Qu[a], Xiang Chen[b,*], Yingquan Zhao[b], and Xiaolin Ju[b]

[a]*School of Mechanical and Electrical Engineering, Jiangsu College of Engineering and Technology, Nantong, 212003, China*
[b]*School of Computer Science and Technology, Nantong University, Nantong, 226000, China*

**Abstract**

Recently, most studies have considered the default value for hyper parameters of the classification methods used by cross-project defect prediction (CPDP) methods. However, in previous studies for within-project defect prediction (WPDP), researchers found that the optimization for hyper parameter helps to improve the performance of software defect prediction models. Moreover, the default value for some hyper parameters in different machine learning libraries (such as Weka, Scikit-learn) may not be consistent. To the best of our knowledge, we first conduct an in-depth analysis for the influence on the performance of CPDP by using hyper parameter optimization. Based on different classification methods, we consider 5 different instance selection based CPDP methods in total. In our empirical studies, we choose 8 projects in AEEEM and Relink datasets as our evaluation subjects, and we use AUC as our model performance measure. Final results show that among these methods, the influence of hyper parameter optimization for 4 methods is non-negligible. Among the 11 hyper parameters considered by these 5 classification methods, the influence of 8 hyper parameters is non-negligible, and these hyper parameters are mainly distributed in support vector machine and k nearest neighbor classification methods. Meanwhile, by analyzing the actual computational cost of hyper parameter optimization, we find that the spent time is within the acceptable range. These empirical results show that in the future CPDP research, the hyper parameter optimization should be considered in experimental design.

*Keywords*: software quality assurance; software defect prediction; cross-project defect prediction; hyper parameter optimization; empirical study

(Submitted on March 2, 2018; Revised on April 16, 2018; Accepted on May 19, 2018)



## 1. Introduction

Defects may be unconsciously introduced in requirement analysis, software design, or software implementation. After deployment, software products with defects may cause huge economic losses to the enterprise and sometimes even threaten people's lives. In the software development life cycle, the later the internal defects are detected, the higher the cost of fixing these defects. Therefore, the software quality assurance department expects to be able to identify as many internal defects as possible before the software product deployment. Software defect prediction [6,12] is one of the feasible methods. In particular, by mining software historical repositories (such as version control system, bug tracking system, developer emails), defect prediction models can be constructed and then used to predict potential defective modules. Then, more testing resources are allocated for these modules, and the quality of software products can be guaranteed. More specifically, researchers designed different novel metrics, which have strong correlation with defects, based on analysis for the complexity of the source code or the characteristics of the development process. For the complexity of the source code, they often consider lines of code [30], McCabe complexity [18], Halstead complexity measures [7], and CK suites [3]. For the characteristics of the development process, they often analyze the characteristic of code changes, the experience of developers, and the dependency of program modules. The granularity of extracted program modules from the version control system can be set as file, class, function, or code change as needed [2,13,14,15,16,23]. Then, researchers used these metrics to measure these extracted program modules and labeled these modules based on the analysis on the bug reports and

* Corresponding author.
*E-mail address*: xchencs@ntu.edu.cn

code change logs [5]. Later, they used a specific classification method (such as naive bayes, support vector machine, or random forest) to train the defect prediction models.

Previous studies mainly focused on within-project defect prediction (WPDP). In particular, researchers used some data of a project to build a defect prediction model and then used the remaining data of the project to evaluate the performance of the built model. However, in the actual software development scenario, the project needing defect prediction (i.e., the target project) may be a new project, or the historical training data of this project is scarce. For this scenario, a simple solution is to build models based on data collected from other projects (i.e., source projects). However, there are large distribution differences between data collected from different projects. The root cause of this problem is that these projects differ in their application domains, programming languages, developers' experience, or characteristics of the development process. The distribution difference of different projects will make it difficult for the above simple solution to obtain a satisfactory prediction performance. Therefore, how to transfer the relevant knowledge from the source project to construct the model for the target project has attracted interest from many researchers. This issue is called cross-project defect prediction (CPDP), and many solutions have been proposed [8,9]. Nowadays, researchers mainly focus on supervised learning based CPDP methods. These methods mainly use instance selection based methods to perform data preprocessing for the source project based on the characteristics of the target project. Then, they use a specific classification method to construct the model. Most of the classification methods used by CPDP have hyper parameters to set. For example, *k* nearest neighbor needs to set the number of nearest neighbors. For the experimental design of most CPDP studies, we cannot obtain the optimal value of hyper parameters in advance. Therefore, we use the default value of these hyper parameters. However, such experimental design has the following issues: (1) In previous studies for WPDP, Tantithamthavorn et al. have found that using hyper parameter optimization can help increase the performance of the trained model [25]. (2) For the implementation of different machine learning libraries, even for the same classification method, the default value of hyper parameter may not be consistent. For example, the number of decision trees is a hyper parameter of random forest classification method. For Weka, the default value of this hyper parameter is 100. While for R, the default value of this hyper parameter is 500 in random forest package. The inconsistency of the default value for hyper parameter will have a certain impact on the validity of previous empirical studies.

Previous research has investigated the influence of hyper parameter optimization for WPDP [25]. However, to the best of our knowledge, no studies have conducted empirical studies to investigate the influence of hyper parameter optimization for CPDP. Based on the above motivation, we mainly analyze the hyper parameter optimization for instance selection based CPDP methods. For these methods, we consider 5 commonly used classification methods and identify 11 hyper parameters in total from these 5 classification methods. In our empirical studies, we use the MultiSearch method provided by Weka to perform hyper parameter optimization. Then, we use AEEEM and Relink datasets as our empirical subjects and use AUC to evaluate the performance of trained models. Final empirical studies show that hyper parameter optimization cannot be neglected for CPDP either, and it should be considered in the experimental design for future CPDP research.

The main contribution of this paper can be summarized as follows:

- To the best of our knowledge, we first investigated the influence of hyper parameter optimization for instance selection based CPDP methods.
- We conducted empirical studies on AEEEM and RELINK datasets and found that hyper parameter optimization cannot be neglected in the experimental design for CPDP.

The rest of this paper is organized as follows: Section 2 introduces related work for cross-project defect prediction. Section 3 gives a simple example to illustrate the motivation of our empirical studies. Section 4 describes the experimental design, including the experimental subjects, performance measure, and the experimental process. Section 5 performs result analysis and threats to validity. Section 6 concludes this paper and discusses some potential future work.

## 2. Related Work

Most of previous studies focused on within-project defect prediction. For this problem, researchers used some data of the project to construct the model, and then used the remaining data to evaluate the performance of the trained model. However, in real development scenario, the project needing defect prediction (i.e., the target project) may be a new project, or this project has few training data. The simple solution is to use the data from other projects (i.e., the source project). However, the application domain, programming language used, or development process characteristics may not the same; the datasets of the source project and the target project cannot satisfy the assumption of independent and identical distribution, and this

poses a research challenge. Therefore, how to transfer the knowledge from the source project to the target project has attracted the attention of researchers, and this problem is called cross-project defect prediction.

Zimmermann et al. [32] conducted large-scale empirical studies on the feasibility of CPDP. They considered many projects from open-source and commercial projects. During 622 cross-project defect prediction scenarios, they found that only in 21 (3.4%) scenarios, CPDP can achieve the satisfactory performance. Then, to further improve the performance of the CPDP, researchers have proposed different effective CPDP methods. Based on the source of datasets and the model construction methods, these CPDP methods can be classified into three categories: supervised learning based, unsupervised learning based, and semi-supervised learning based. In particular, supervised learning based methods constructed models based on the labelled program modules from the candidate source projects. Based on whether they used the same metrics, these methods can be further classified into heterogeneous methods and homogeneous methods. Here, homogeneous methods are based on instance selection [27], instance weight setting [17], and feature mapping [20], while heterogeneous methods perform feature mapping between the source project and the target project based on the distribution similarity analysis of different metrics [10,21]. Unsupervised learning based methods attempt to predict directly on the datasets of the target project. The assumption of these methods is that the value of defective modules often has a higher tendency than non-defective modules. The classical methods include CLA and CLAMI proposed by Nam et al. [22] and spectral clustering based methods by Zhang et al. [31]. Semi-supervised learning based methods use the datasets from the source project and some datasets from the target project. These methods aimed to choose a few representative modules from the target project to improve the performance of the trained models. Classical methods include HYDRA proposed by Xia et al. [29].

Tantithamthavorn [24] analyzed the most influential factors in the experimental design of empirical studies for software defect prediction. They mainly considered noise issue in the datasets, hyper parameter optimization, and model evaluation methods. Here, we mainly analyze the issues when considering the default value for hyper parameters. For example, Jiang et al. [11] and Tosun et al. [26] found that some methods will achieve unsatisfactory performance when considering default value of hyper parameters. Mende et al. [19] found that changing the value of the hyper parameter will influence the performance of later trained model. Recently, Tantithamthavorn et al. [25] conducted an in-depth analysis for this issue. They considered 30 different classification methods from 11 different categories and found that at least 26 methods have one hyper parameter to set. They conducted a large-scale empirical study for WPDP and found that the influence of hyper parameter optimization cannot be neglected for 16 classification methods. Moreover, using hyper parameter optimization can increase the performance by up to 40 percentage points.

## 3. A Motivation Example

We use a simple motivation example to analyze the influence of hyper parameter optimization for cross-project defect prediction. In this example, we use EQ in AEEEM dataset as the source project, and use JDT in AEEEM dataset as the target project. For CPDP, we use Burak filter [27] to perform instance selection, then use *k* nearest neighbour as the classification method. This classification method has two hyper parameters: KNN and distanceWeighting. The default value and the optimized value of these two hyper parameters can be found in Table 1. The detail of cross-project defect prediction method and the meaning of hyper parameters of the k nearest neighbour can be found in Section 4. The final prediction result on the target project (based on AUC performance measure) is shown in Table 1. From this result, we can find that using hyper parameter optimization, the performance of CPDP can increase by 15.4 percentage points. By this motivation example, we find that hyper parameter optimization can have a certain influence on CPDP. Then, we will conduct large-scale empirical studies on this issue by considering more subjects and more CPDP methods.

Table 1. Influence of Hyper Parameter Optimization for CPDP (EQ is set as the source project and JDT is set as the target project)

| | **KNN** | **distanceWeighting** | **AUC** |
|---|---|---|---|
| Default Value | 1 | No distance weighting | 0.596 |
| Optimized Value | 10 | Weight by 1/distance | 0.750 |

## 4. Experimental Design

To investigate the influence of hyper parameter optimization for CPDP, we design the following three research questions:

- **RQ1**: What is the influence of hyper parameter optimization on cross-project defect prediction?
- **RQ2**: What is the influence of hyper parameters on cross-project defect prediction?
- **RQ3**: What is the computational cost of hyper parameter optimization?

### *4.1. Experimental Subjects*

In our empirical studies, we consider AEEEM and Relink datasets. These datasets have high quality and have been widely used in previous studies [8,9]. Moreover, these datasets are gathered by different research groups, so the generality of our empirical results can be guaranteed.

AEEEM is gathered by D'Ambros et al. [1]. The granularity of the extracted program module is set as class, and 61 metrics are considered. These metrics consider not only code complexity but also the characteristics of the development process, such as code modification characteristic, historical defect detection information, and module complexity.

Relink is gathered by Wu et al. [28]. The granularity of the extracted program module is set as the file, and 26 metrics are considered. These metrics only consider the complexity of code, and the values of these metrics are gathered by using the understand tool (http://www.scitools.com). Wu et al. further improved the quality of Relink datasets in the manual manner.

The characteristics of these datasets are shown in Table 2 and Table 3 respectively, including project name (abbreviation), the granularity of the extracted program modules, the number of considered metrics, the number of program modules, and the number of defective modules.

Table 2. The Characteristic of AEEEM Dataset

| Project Name | Granularity | Metrics | Modules | Defective Modules |
|---|---|---|---|---|
| Apache | Class | 26 | 194 | 98 |
| Safe | Class | 26 | 56 | 22 |
| ZXing | Class | 26 | 399 | 118 |

Table 3. The Characteristic of Relink Dataset

| Project Name | Granularity | Metrics | Modules | Defective Modules |
|---|---|---|---|---|
| Equinox (EQ) | File | 61 | 325 | 129 |
| Eclipse JDT Core (JDT) | File | 61 | 997 | 206 |
| Apache Lucene (LC) | File | 61 | 399 | 691 |
| Mylyn (ML) | File | 61 | 245 | 245 |
| Eclipse PDE UI (PDE) | File | 61 | 1492 | 209 |

### *4.2. Performance Measure*

Traditional performance measures (such as precision, recall, F-measure) for software defect prediction are threshold-dependent. In particular, if the prediction value for a module by using the trained module is larger than the threshold, then the module is predicted as the defective module. Otherwise, it is predicted as the non-defective module. Therefore, these measures are dependent on the threshold. Nowadays, most researchers set the value of threshold as 0.5 by default. In our empirical studies, we use AUC (area under ROC curve) to evaluate the performance of the trained model. ROC curve evaluates the performance when considering different thresholds. In particular, in the ROC curve, the *x*-axis denotes the false positive rate, while the *y*-axis denotes the true positive rate. For different thresholds, the value of the trained model corresponds to the false positive rate and the true positive rate (i.e., a coordinate point). Connecting all the coordinate points can generate a ROC curve. The closer the value of AUC is to 1, the better the performance of the trained model. When comparing to threshold-dependent measures (such as accuracy, precision, recall, F-measure), AUC has such advantages. For example, it cannot be influenced by the class imbalance problem, which is common in gathered datasets of software defect prediction. Meanwhile, it does not need the setting of threshold value. Therefore, AUC is widely used in current software defect prediction research [8,9].

### *4.3. Experimental Process*

The total process of our empirical study can be found in Figure 1. Given the source project and the target project, we first conduct cross-project defect prediction models based on the source project. The method includes two phases: instance selection phase and classification phase. In the instance selection phase, we consider Burak filter [27]. In the classification phase, we consider two choices: using hyper parameter optimization or using the default value for hyper parameters. Based on these two choices, we can generate two different models. Then, we can get the performance of the target project based on these two different models. In the rest of this subsection, we will give the details of the instance selection method, considered classification methods, and hyper parameter optimization.

1) Instance Selection Phase: By using instance selection, we can choose relevant instances from the source project by

considering the characteristic of the target project, and we can then improve the performance of CPDP. In this paper, we consider a classical instance selection method (i.e., Burak filter [27]). In particular, the Burak filter was proposed by Turhan et al. [27]. This method first computes the Euclidean distance between instances from the source project and the target project. Then, for each instance in the target project, it will choose the top $k$ instances (the value of $k$ is set as 10 in our empirical studies) and add them to the final training set. Supposing the target project has $N$ instances, the number of final chosen instances will be $k \times N$. Here, an instance may be chosen multiple times, but the final training set only includes this instance once.

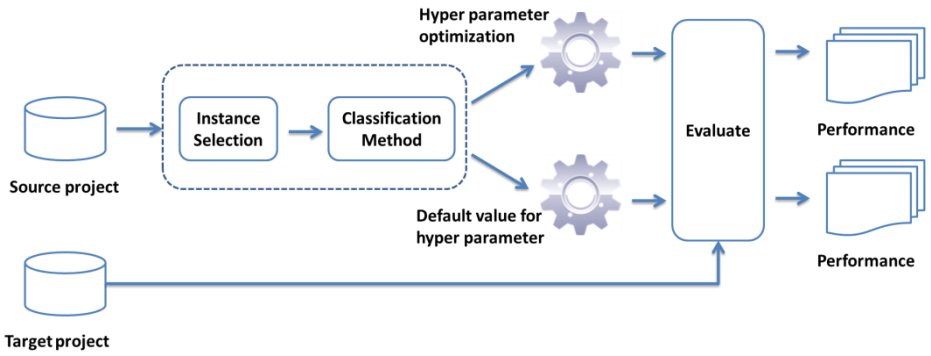


Figure 1. The Total Process of Experiments

2) Classification Phase: In our empirical studies, we consider 5 different classification methods, such as $k$ nearest neighbors, naive bayes, decision tree, random forest, and support vector machine. Some widely used classification methods were not considered because these methods do not have hyper parameters (such as Logistic regression).

$k$ nearest neighbor (IBK) has two hyper parameters:

- IBK-HP1: This hyper parameter denotes the number of nearest neighbors and refers to KNN variable in Weka. The candidate value set is {1,2,⋯,10}, and the default value is 1.
- IBK-HP2: This hyper parameter denotes whether setting weight by using the distance and refers to distanceWeighting variable in Weka. The candidate value set is {no distance weighting, weight by 1/distance, weight by 1-distance}, and the default value is no distance weighting.

Naive bayes (NB) has two hyper parameters:

- NB-HP1: This hyper parameter denotes whether the normal distribution assumption is considered when using kernel estimator for the numerical features and refers to use KernelEstimator variable in Weka. The candidate value set is {true, false}, and the default value is false.
- NB-HP2: This hyper parameter denotes whether numerical features are converted into nominal features by using supervised based discritization method. The candidate value set is {true, false}, and the default value is false.

Decision tree (J48) has 4 hyper parameters:

- J48-HP1: This hyper parameter denotes the confidence factor used for pruning (smaller values incur more pruning) and refers to confidenceFactor variable in Weka. The candidate value set is {0.1,0.15, ⋯,0.5}, and the default value is 0.25.
- J48-HP2: This hyper parameter denotes the minimum number of instances per leaf and refers to minNumObj variable in Weka. The candidate value set is {1,2, ⋯,64}, and the default value is 2.
- J48-HP3: This hyper parameter denotes whether the subtree raising operation is considered when performing pruning and refers to subtreeRaising variable in Weka. The candidate value set is {true, false}, and the default value is false.
- J48-HP4: This hyper parameter denotes whether to use binary splits on nominal attributes when building the trees and refers to binarySplits variable in Weka. The candidate value set is {true, false}, and the default value is false.

Random forest (RF) has 2 hyper parameters:

- RF-HP1: This hyper parameter denotes whether random seeds are used and refers to seed variable in Weka. The candidate value set is {1,2, ⋯,5}, and the default value is 1.
- RF-HP2: This hyper parameter denotes the number of decision trees and refers to numIterations in Weka. The default value set is {100,200, ⋯,400 }, and the default value is 100.

Support vector machine (SVM) has only one hyper parameter. In our empirical studies, we set kernel function as RBFKernel.

- SVM-HP1: This hyper parameter denotes the hyper parameter for RBFKernel and refers to kernel gamma in Weka. The default value set is {0.0001, 0.001, 0.01, 0.1, 1, 10}, and the default value is 0.1.

3) Hyper Parameter Optimization: Commonly used machine learning libraries support hyper parameter optimization, such as Caret package in R, MultiSearch in Weka, and GridSearch in Scikit-learn. In this paper, we utilize MultiSearch in Weka. In particular, it first identifies candidate values for each hyper parameter. Then, it will generate all possible hyper parameter solutions. Supposing a classification method has two hyper parameters and each parameter has 5 candidate values, then there are 25 feasible hyper parameter solutions for this method. To improve the computational efficiency, MultiSearch uses 2-fold cross-validation to identify the optimal hyper parameter solution s, and it then searches for better solutions around the solution s by hill climbing and using 10-fold cross-validation.

## 5. Analysis of Results

### *5.1. Analysis of Results for RQ1*

In this RQ, we mainly analyze the influence of hyper parameter optimization on the CPDP. For a specific CPDP scenario (given a source project and target project), the performance of using default value of hyper parameters is $x$ and the performance of using hyper parameter optimization is $y$, and then we can use $y$-$x$ to denote the performance difference of these two methods. We use a box plot to show the distribution of performance difference in 26 CPDP scenarios in total (i.e., 6 scenarios for AEEEM dataset and 20 scenarios for Relink dataset). To further conduct a quantitative analysis for the magnitude of the performance difference, we use Cohen's $d$ effect size [4]. We assume that the average value and standard deviation for the method $A$ is $X_A$ and $S_A$ for all the CPDP scenarios, and the average value and standard deviation for the method $B$ is $X_B$ and $S_B$. Then, the formula of Cohen's $d$ effect size is $|X_B - X_A|/\sqrt{S_A^2 + S_B^2/2}$. The effect size and its value range are shown in Table 4. Final performance difference distribution and effect size are shown in Figure 2.

Table 4. Cohen's *d* effect size and its value range

| Effect Size | Value Range |
|---|---|
| Negligible | d<=0.2 |
| Small | 0.2<d<=0.5 |
| Medium | 0.5<d<=0.8 |
| Large | 0.8<d |

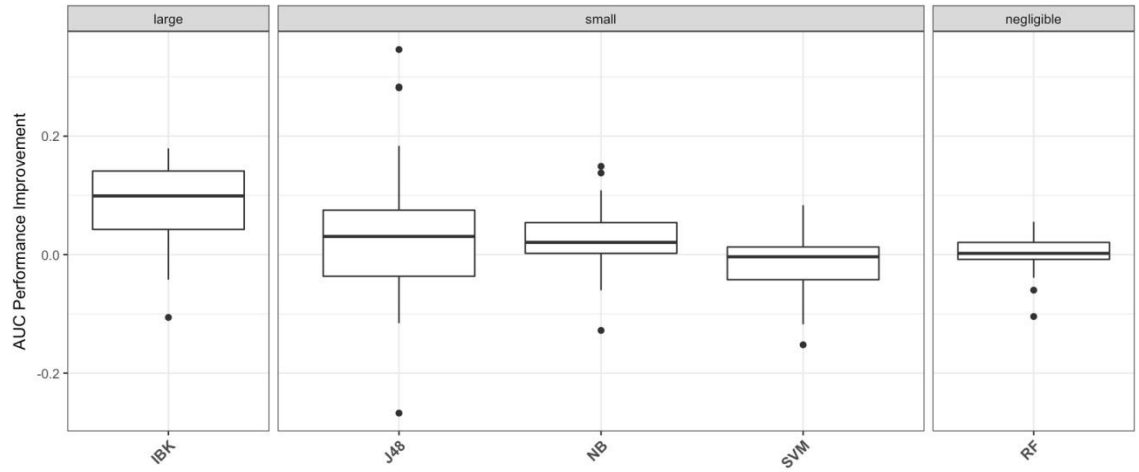


Figure 2. The performance difference distribution and its Cohen's d effect size for each classification method

On average, the performances of most CPDP methods are influenced by using hyper parameter optimization. Based on Cohen's *d* effect size, 4 of 5 CPDP methods (i.e., 80%) cannot be neglected (i.e., Cohen's *d* value is larger than 0.2). The effect size of IBK is large (value is 1.15), while for J48, NB, and SVM, the effect size is small (value is 0.38, 0.32, and 0.30 respectively). For RF, the effect size is negligible (value is 0.02). By using hyper parameter optimization, the performance can be increased by 34.63 percentage points at most. Moreover, hyper parameter optimization is most effective for IBK and can increase the performance by 8.6 percentage points on average.

### *5.2. Analysis of Results for RQ2*

In this RQ, we mainly identify which hyper parameters have the most influence on the performance of CPDP. Therefore, in the later empirical studies, researchers should pay more attention on them.

In this paper, we consider 11 hyper parameters of 5 classification methods. The meaning of these hyper parameters can be found in Section 4. Given a classification method, we first identify the default value and the optimized value based on the analysis for RQ1. We then compute the performance (i.e., *x*) of using the default value and the performance of using the optimized value (i.e., *y*), and we use (*y*-*x*) to denote the performance difference. The final result is shown in Figure 3.

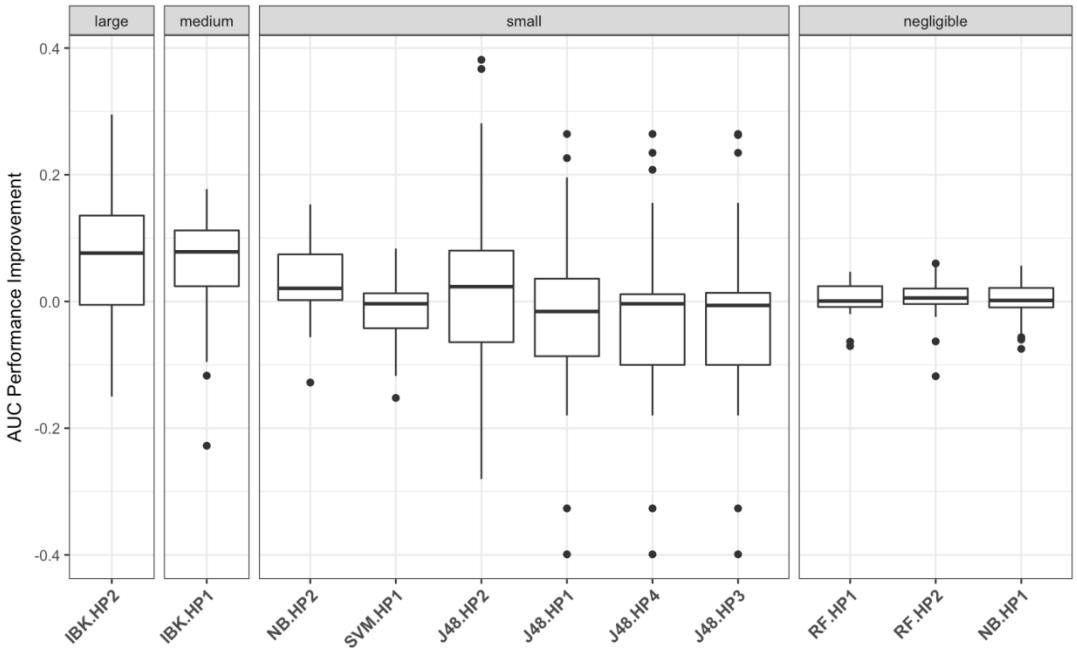


Figure 3. The performance difference distribution and its Cohen's *d* effect size for each hyper parameter

### *5.3. Analysis of Results for RQ3*

In this RQ, we mainly analyze the computational cost for using hyper parameter optimization. The environment of our empirical studies is: Win 10 operation system, 16G memory and CPU E5-2670. Final results are shown in Table 5, including minimum value (Min), maximal value (Max), average value (Average), and standard deviation (SD). It is not hard to find that using hyper parameter optimization should at least spend 1 second and use 3 minutes (based on RF) at most. Therefore, the computational cost of using hyper parameter can be acceptable.

Table 5. Computation Cost of Using Hyper Parameter Optimization (Unit: Second)

| Classification Method | Min | Max | Average | SD |
|---|---|---|---|---|
| IBK | 0.062 | 26.844 | 5.937 | 7.091 |
| J48 | 1.342 | 126.03 | 42.346 | 38.593 |
| NB | 0.051 | 0.671 | 0.268 | 0.207 |
| RF | 0.94 | 154.35 | 36.605 | 43.917 |
| SVM | 0.134 | 2.684 | 0.681 | 0.747 |

### *5.4. Threats to Validity*

Threats to construct validity consider whether the performance metrics used in the empirical studies reflect the real-world situation. In our experimental study, we mainly use the AUC metric, which is commonly used in current software defect prediction, to evaluate the performance of the trained model. Threats to internal validity are mainly concerned with the uncontrolled internal factors that might have influence on the experimental results. The internal threat is the potential faults during our method implementation. To reduce this threat, we use test cases to verify the correctness of our implementation, and we use mature third-party libraries, such as packages from Weka. Threats to external validity consider whether the observed experimental results can be generalized to other subjects. To guarantee the representative of our empirical subjects, we chose AEEEM and Relink, which have been widely used in previous software defect prediction research studies [8,9].

## 6. Conclusions

In this paper, we mainly analyzed the hyper parameter optimization for CPDP. Empirical results show that the influence of hyper parameter optimization cannot be neglected. Therefore, we suggest that in the future research of CPDP, researchers should consider the influence of hyper parameter optimization during the experimental design.

There are some future works needing further investigation: (1) Our empirical results need verification by more open source projects and commercial projects. (2) We need to investigate more classification methods and identify their hyper parameters. (3) We need to investigate the influence of hyper parameter optimization on other new proposed CPDP methods.

## Acknowledgments

This work has been supported by key laboratory of distributed generation and microgrid technology in Nantong (CP12015007), natural science research project of Jiangsu College of Engineering and Technology (GYKY/2016/15).

## References

1. M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proceedings of International Conference on Mining Software Repositories*, 2010, pp. 31–41.
2. X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "Multi: Multi-objective effort-aware just-in-time software defect prediction," *Information & Software Technology*, vol. 93, pp. 1–13, 2018.
3. S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp.476–493, 1994.
4. J. Cohen, "Statistical power analysis for the behavioral sciences," *Journal of the American Statistical Association*, vol. 2, no. 4, pp. 19–74,1988.
5. D. A. D. Costa, S. Mcintosh, W. Shang, U. Kulesza, R. Coelho, and A. Hassan, "A framework for evaluating the results of the szz approach for identifying bug-introducing changes," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2016.
6. T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp.1276–1304, 2012.
7. M. H. Halstead, Elements of Software Science (Operating and Programming Systems Series). New York, NY, USA: Elsevier Science Inc., 1977.
8. S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2017.
9. S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2017.
10. X. Jing, F. Wu, X. Dong, F. Qi, and B. Xu, "Heterogeneous cross-company defect prediction by unified metric representation and CCA based transfer learning," in *Proceedings of the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2015, pp. 496–507.
11. Y. Jiang, B. Cukic, and T. Menzies, "Can data transformation help in the detection of fault-prone modules?" in *Proceedings of The Workshop on Defects in Large Software Systems*, 2008, pp. 16–20.
12. Y. Kamei and E. Shihab, "Defect prediction: Accomplishments and future challenges," in *Proceedings of International Conference on Software Analysis, Evolution, and Reengineering*, 2016, pp. 33–45.
13. T. Lee, J. Nam, D. Han, S. Kim, and H. P. In, "Developer micro interaction metrics for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 42, no. 11, pp. 1015–1035, 2016.
14. S. Liu, X. Chen, W. Liu, J. Chen, Q. Gu, and D. Chen, "Fecar: A feature selection framework for software defect prediction," in *Proceedings of Annual Computer Software and Applications Conference*, 2014, pp. 426–435.
15. W. Liu, S. Liu, Q. Gu, J. Chen, X. Chen, and D. Chen, "Empirical studies of a two-stage data preprocessing approach for software fault prediction," *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 38–53,2016.

16. W. Liu, S. Liu, Q. Gu, X. Chen, and D. Chen, "Fecs: A cluster based feature selection method for software fault prediction with noises," in *Proceedings of Annual Computer Software and Applications Conference*,2015, pp. 276–281.
17. Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information & Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.
18. T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.
19. T. Mende, "Replication of defect prediction studies: problems, pitfalls and recommendations," in *Proceedings of International Conference on Predictive MODELS in Software Engineering*, 2010, p. 5.
20. J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings of the International Conference on Software Engineering*, 2013, pp. 382–391.
21. J. Nam and S. Kim, "Heterogeneous defect prediction," in *Proceedings of the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2015, pp. 508–519.
22. J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets," in *Proceedings of International Conference on Automated Software Engineering*, 2015, pp. 452–463.
23. C. Ni, W. S. Liu, X. Chen, Q. Gu, D. X. Chen, and Q. G. Huang, "A cluster based feature selection method for cross-project software defect prediction," *Journal of Computer Science and Technology*, vol. 32, no. 6, pp. 1090–1107, 2017.
24. C. Tantithamthavorn, "Towards a better understanding of the impact of experimental components on defect prediction modelling," in *Proceedings of International Conference on Software Engineering*, 2017, pp.867–870.
25. C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the International Conference on Software Engineering*, 2016, pp. 321–332.
26. A. Tosun and A. Bener, "Reducing false alarms in software defect prediction by decision threshold optimization." in *Proceedings of International Symposium on Empirical Software Engineering and Measurement*,2009, pp. 477–480.
27. B. Turhan, T. Menzies, A. B. Bener, and J. D. Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.
28. R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: Recovering links between bugs and changes," in *Proceedings of the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2011, pp. 15–25.
29. X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *IEEE Transactions on Software Engineering*, vol. 42, no. 10, pp. 977–998,2016.
30. H. Zhang, "An investigation of the relationships between lines of code and defects," in *Proceedings of International Conference on Software Maintenance*, 2009, pp. 274–283.
31. F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *Proceedings of the International Conference on Software Engineering*, 2016,pp. 309–320.
32. T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs.domain vs. process," in *Proceedings of Joint Meeting of the European Software Engineering Conference and the International Symposium on Foundations of Software Engineering*, 2009, pp. 91–100.

**Yubin Qu** received a B.S. degree and M.Sc. degree in Computer Science and Technology from Henan Polytechnic University in China in 2004 and 2008. He lectures at the School of Mechanical and Electrical Engineering at the Jiangsu College of Engineering and Technology. His main research interests are software maintenance, software testing, and machine learning.

**Xiang Chen** received a B.Sc. degree in the School of Management from Xi'an Jiaotong University in China in 2002. He then received M.Sc. and Ph.D. degrees in Computer Science from Nanjing University in China in 2008 and 2011, respectively. He is an associate professor in the Department of Computer Science and Technology at Nantong University. His main research interests are software maintenance and software testing, including security vulnerability prediction, software defect prediction, combinatorial testing, regression testing, and fault localization. He has published over 40 papers in referred journals or conferences, including IST, JSS, TRel, SQJ, JCST, COMPSAC, APSEC, and SAC.

**Yingquan Zhao** received a B.S. degree in Computer Science from Nantong University in China in 2018. He is currently pursuing a Master's degree in the School of Computer Software at Tianjin University. His research interests include security vulnerability prediction, software defect prediction, and data mining.

**Xiaolin Ju** received a Ph.D. degree in Computer Science and Technology from the China University of Mining and Technology in 2014. He is currently an associated professor at Nantong University. His research interests include software testing and analysis, program debugging, and fault localization.