Understanding the Automated Parameter Optimization on Transfer Learning for Cross-Project Defect Prediction: An Empirical Study

ABSTRACT

Data-driven defect prediction has become increasingly important in software engineering process. Since it is not uncommon that data from a software project is insufficient for training a reliable defect prediction model, transfer learning that borrows data/konwledge from other projects to facilitate the model building at the current project, namely Cross-Project Defect Prediction (CPDP), is naturally plausible. Most CPDP techniques involve two major steps, *i.e.,* transfer learning and classification, each of which has at least one parameter to be tuned to achieve their optimal performance. This practice fits well with the purpose of automated parameters optimization. However, there is a lack of thorough understanding about what are the impacts of automated parameters optimization on various CPDP techniques. In this paper, we present the first empirical study that looks into such impacts on 62 CPDP techniques, 13 of which are chosen from the existing CPDP literature while the other 49 ones have not been explored before. We build defect prediction models over 20 real-world software projects that are of different scales and characteristics. Our findings demonstrate that: (1) Automated parameter optimization substantially improves the defect prediction performance of 77% CPDP techniques with a manageable computational cost. Thus more efforts on this aspect are required in future CPDP studies. (2) Transfer learning is of ultimate importance in CPDP. Given a tight computational budget, it is more cost-effective to focus on optimizing the parameter configuration of transfer learning algorithms (3) The research on CPDP is far from mature where it is 'not difficult' to find a better alternative by making a combination of existing transfer learning and classification techniques. This finding provides important insights about the future design of CPDP techniques.

CCS CONCEPTS

• **Software and its engineering → Software creation and management**; **Software defect analysis**.

KEYWORDS

Cross-project defect prediction, transfer learning, classification techniques, automated parameter optimization

1 INTRODUCTION

According to the latest *Annual Software Fail Watch* report from Tricentis[1], software defects/failures affected over 3.7 billion people and caused \$1.7 trillion in lost revenue. In practice, stakeholders usually have limited software quality assurance resources to maintain a software project. Identifying high defect-prone software modules (*e.g.,* files, classes or functions) by using advanced statistical and/or machine learning techniques, can be very helpful for software engineers and project managers to prioritize their actions in the software development life cycle and address those defects.

It is well known that a defect prediction model works well if it is trained with a sufficient amount of data [19]. However, it is controversial to obtain adequate data (or even having no data at all) in practice, especially when developing a brand new project or in a small company. By leveraging the prevalent transfer learning techniques [51], cross-project defect prediction (CPDP) [8] has become increasingly popular as an effective way to deal with the shortage of training data [75]. Generally speaking, its basic idea is to leverage the data from other projects (*i.e.,* source domain projects) to build the model and apply it to predict the defects in the current one (*i.e.,* target domain project).

Defect prediction models usually come with configurable parameters (87% prevalent classification techniques are with at least one parameter [64, 65]), the settings of which largely influence the prediction accuracy when encountering unseen software projects [40, 41]. It is not difficult to envisage that the optimal settings for those parameters are problem dependent and are unknown beforehand. Without specific domain expertise, software engineers often train their defect prediction models with off-the-shelf parameters suggested in their original references. This practice may undermine the performance of defect prediction models [19] and be adverse to the research reproducibility of defect prediction studies [44, 45]. Recently, Tantithamthavorn *et al.* [64, 65] have empirically shown the effectiveness and importance of automated parameter optimization for improving the performance and stability of many prevalent classification techniques for defect prediction with manageable additional computational cost.

When considering CPDP, defect prediction become more complicated. To transfer knowledge from the source to the target domain, prevalent transfer learning techniques naturally bring additional configurable parameters. According to our preliminary literature study, 28 out of 33 most widely used CPDP techniques (85%) require at least one parameter to setup in the transfer learning (or as known as domain adaptation) stage. This complication inevitably brings more challenges to the parameter optimization due to the further explosion of the parameter space, such as the extra difficulty of finding the optimal configuration and the increased computational cost for evaluating the model during optimization. Although hyper-parameter optimization (also known as automated machine learning) has been one of the hottest topics in the machine learning community [30], to the best of our knowledge, little research have been conducted in the context of transfer learning.

Bearing these considerations in mind, in this paper, we seek to better understand how automated parameter optimization of transfer learning models can impact the performance in CPDP through a systematic and empirical study. In doing so, we aim to

[1]https://www.tricentis.com/resources/software-fail-watch-5th-edition/

gain constructive insights based on which one can further advance this particular research area. To this end, the first research question (RQ) we wish to answer is:

> **RQ1**: ***How much benefit of automated parameter optimization can bring to the performance of defect prediction models in the context of CPDP?***

Answering **RQ1** is not straightforward, because transfer learning and classification are two intertwined parts in a CPDP model. Both of them have configurable parameters that control the characteristics of the CPDP model they produce. Therefore, the automated parameter optimization can be conducted by using three possible types of methods, all of which need to be studied for **RQ1**:

- `Type-I`: Naturally, it makes the most sense to optimize the parameters of both transfer learning and classification simultaneously. However, due to the large parameter space, it might be challenging to find the optimal configuration within a limited budget of computational cost.
- `Type-II`: For the sake of taking the best use of computational cost, alternatively, parameters optimization may be conducted on one part of a CPDP model, *i.e.,* either the transfer learning (denoted as `Type-II-1`) or the classification (denoted as `Type-II-2`), at a time; while the other part is trained by using the default parameters. In this way, the parameter space is reduced, and so does the necessary computational cost. However, this might not necessarily imply an undermined performance. For example, if transfer learning is the determinant part of a CPDP model in general, then optimizing it alone is expected to have at least the same level of result as optimizing both transfer learning and classification together (*i.e.,* `Type I`) while causing much less computational cost.
- `Type-III`: Finally, the automated parameter optimization can also be conducted in a sequential manner where the transfer learning part is optimized before the classification model[2]. In particular, each part is allocated half of the total budget of computational cost. In this method, although the total computational cost is exactly the same as that of `Type I`, the parameter space is further constrained, which enables more focused search behaviors.

The above also motivates our second RQ, in which we ask:

> **RQ2**: ***What is the most cost effective type of automated parameter optimization given a limited amount of computational cost?***

Investigating **RQ1** and **RQ2** would inevitably require us to go through a plethora of transfer learning and classification techniques proposed in the machine learning literature [51]. During the process, we found that the transfer learning and classification techniques in existing CPDP models are either selected in a problem-specific or ad-hoc manner. Little is known about the versatility of their combinations across various CPDP tasks with different characteristics. Because of such, our final RQ seeks to understand:

[2]The parameters of a classification model is set as default values when optimizing the transfer learning part.

> **RQ3**: ***Whether the state-of-the-art combinations of transfer learning and classification techniques can indeed represent the generally optimal settings?***

To address the above RQs, we apply `Hyperopt` [7], an off-the-shelf hyper-parameter optimization toolkit[3], as the fundamental optimizer on the CPDP models considered in our empirical study. Comprehensive and empirical study is conducted on 62 combinations of the transfer learning algorithms and classifiers, leading to a total of 37 different parameters to be tuned, and using 20 datasets from real-world open source software projects. In a nutshell, our findings answer the RQs as below:

- **To RQ1:** Automated parameter optimization can substantially improve the CPDP techniques considered in our empirical study. In particular, 77% of the improvements have been classified as *huge* according to the Cohen's d effect size.
- **To RQ2:** Transfer learning is the most determinant part in CPDP while optimizing its parameters alone can achieve better CPDP performance than the other types of automated parameter optimization.
- **To RQ3:** No. Some 'newly' developed CPDP techniques, with under-explored combinations of transfer learning and classification techniques, can achieve better (or at least similar) performance than those state-of-the-art CPDP techniques.

Drawing on those answers, our empirical study, for the first time, provides new insights that help to further advance this field of research[4]:

- Automated parameter optimization can indeed provide significant improvement to the CPDP model, within which optimizing the parameters of transfer learning is the most determinant part. This observation suggests that future research on the optimizer can primarily focus on this aspect in the design and hence prevent wasting efforts on other methods that provide no better performance but generating extra computational cost only.
- The state-of-the-art combinations of transfer learning and classifier are far from being optimal, implying that the selection of combination is at least as important as the parameter tuning. As a result, future work should target a whole portfolio of optimization, tuning not only the parameters, but also the algorithmic components, i.e., the selection of appropriate transfer learning and classifier pair, of a CPDP model.

The rest of this paper is organized as follows. Section 2 provides the methodology used to conduct our empirical study. Section 3 present and analyze the experimental results. Thereafter, the results and threads to validity are discussed in Section 5. At the end, Section 6 concludes the findings in this paper and provides some potential future directions.

2 THE EMPIRICAL STUDY METHODOLOGY

This section elaborates the methodology and experimental setup of our empirical study, including the dataset selection, the working

[3]http://hyperopt.github.io/hyperopt/

[4]To enable full reproducibility, all the experimental data and code for our empirical study can be found in the attached supplementary files.

principle of Hyperopt, the system architecture of automated parameter optimization for CPDP model building and the performance metric used to evaluate the performance of a CPDP model.

2.1 Dataset Selection

In our empirical study, we use the following six criteria to select the datasets for CPDP model building.

(1) To promote the research reproducibility of our experiments, we choose datasets hosted in public repositories.
(2) To mitigate potential conclusion bias, the datasets are chosen from various corpora and domains. More importantly, the shortlisted datasets in our empirical study have been widely used and tested in the CPDP literature.
(3) If the dataset has more than one version, only the latest version is used to train a CPDP model. This is because different versions of the same project share a lot of similarities which simplify transfer learning tasks.
(4) To avoid overfiting a CPDP model, the datasets should have enough instances for model training.
(5) To promote the robustness of our experiments, it is better to inspect the datasets to rule out data that are either repeated or having missing values.
(6) To resemble real-world scenarios, it is better to choose datasets from open source projects provided by iconic companies.

According to the first two criteria and some recent survey papers on the CPDP research [25, 27, 28, 75], we shortlist five publicly available datasets (*i.e.,* JURECZKO, NASA, SOFTLAB, AEEEM, ReLink). Note that these datasets are from different domains and have been frequently used in the literature. To meet the fourth criterion, we further rule out SOFTLAB while NASA is also not considered in our experiments due to its relatively poor data quality [62] according to the fifth criterion. In summary, the datasets used in our experiments consist of 20 open source projects with 10,952 instances. The characteristics of each dataset are summarized as follows:

- AEEEM [15]: This dataset contains 5 open source projects with 5,371 instances. In particular, each instance has 61 metrics with two different types, including static and process metrics like the entropy of code changes and source code chorn.
- ReLink [71]: This dataset consists of 3 open source projects with 649 instances. In particular, each instance is with 26 static metrics. Note that the defect labels are further manually verified after being generated from source code management system commit comments.
- JURECZKO [31]: This dataset originally consists of 92 released products collected from open source, proprietary and academic projects. To meet the first criterion, those proprietary projects are ruled out from our consideration. To meet the last criterion, the projects merely for academic use are excluded from JURECZKO. Moreover, since the projects in JURECZKO have been updated more than one time, according to the third criterion, only the latest version is considered in our experiments. At the end, we choose 12 open source projects with 4,932 instances in our empirical study.

2.2 Hyperopt for Automated Parameter Optimization

Hyperopt[5] is a Python library that provides algorithms and the software infrastructure to optimize hyperparameters of machine learning algorithms. Hyperopt uses its basic optimization driver hyperopt.fmin to optimize the parameter configurations of the CPDP techniques considered in our empirical study. Using Hyperopt requires three steps.

- **Define an objective function:** As suggested in [6], Hyperopt provides a flexible level of complexity when specifying an objective function that takes inputs and returns a loss users want to minimize. In our experiments, the inputs are parameters associated with the transfer learning and classification techniques as shown in the third column of Table 1 and Table 2 respectively.
- **Define a configuration space:** The configuration space in the context of Hyperopt describes the domain over which users want to search. The last column of Table 1 and Table 2 list the configuration space of the corresponding parameter.
- **Choose an optimization algorithm:** Hyperopt provides two alternatives, *i.e.,* random search [5] and Tree-of-Parzen-Estimators (TPE) algorithm [4], to carry out the parameter optimization. In our experiments, we choose to use the TPE algorithm because the sequential model-based optimization has been recognized as one of the most effective hyperparameter optimization methods in the auto-machine learning literature [18]. In practice, Hyperopt provides a simple interface to deploy the TPE algorithm where users only need to pass algo=hyperopt.tpe.suggest as a keyword argument to hyperopt.fmin.

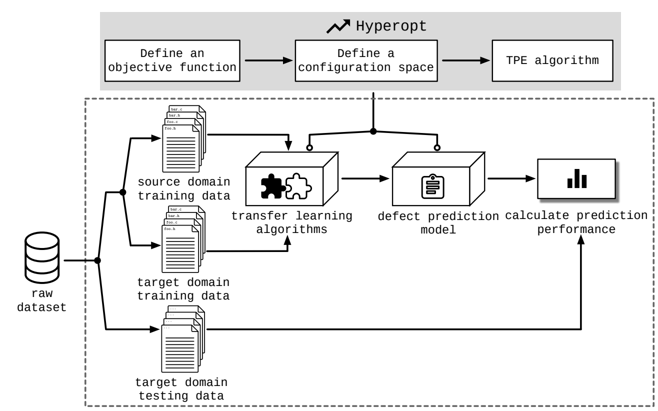

Figure 1: The architecture of automated parameter optimization on CPDP model by using Hyperopt.

2.3 Architecture for Optimizing CPDP Model

Figure 1 shows the architecture of using Hyperopt to optimize the performance of CPDP models. The implementation details of our empirical study are given below.

[5] http://hyperopt.github.io/hyperopt/

Table 1: The parameters of studied transfer techniques

Transfer techniques	Parameters		
	Name	**Description**	**Range**
Bruakfilter	k	The number of neighbors to each point (default=10) [N]	[1, 100]
DS	topN	The number of closest training sets (default=5) [N]	[1, 15]
	FSS	The ratio of unstable features filtered out (default=0.2) [R]	[0.1, 0.5]
DSBF	Toprank	The number assigned to 1 when performing feature reduction (default=1) [N]	[1, 10]
	k	The number of neighbors to each point (default=25) [N]	[1, 100]
TCA	kernel	The type of kernel (default='linear') [C]	{'primal', 'linear', 'rbf', 'sam'}
	dimension	The dimension after tranforing (default=5) [N]	[5, max(N_source, N_target)]
	lamb	Lambda value in equation (default=1) [R]	[1e-6, 1e2]
	gamma	Kernel bandwidth for 'rbf' kernel (default=1) [R]	[1e-5, 1e2]
DBSCANfilter	eps	The maximum distance between two samples for one to be considered as in the neighborhood of the other (default=1.0) [R]	[0.1, 1e2]
	min_samples	The number of samples (or total weight) in a neighborhood for a point to be considered as a core point (default=10) [N]	[1, 100]
Universal	p-value	The associated p-value for statistical test (default=0.05) [R]	[0.01, 0.1]
	quantifyType	The type of quantifying the difference between distributions (default='cliff') [C]	{'cliff', 'cohen'}
DTB	k	The number of neighbors to each point (default=10) [N]	[1, 50]
	T	The maximum number of iterations (default=20) [N]	[5, 30]
Peterfilter	r	The number of points in each cluster (default=10) [N]	[1, 100]

[N] An integer value from range
[R] A real value from range
[C] A choice from categories

- As shown in Figure 1, given a raw dataset with N projects, $N-1$ of which are used as the source domain data while the other project is used as the target domain data. In particular, all source domain data are used for training; part of the target domain data is used to train the CPDP model while the remaining ones are used for the testing purpose. To mitigate a potentially biased conclusion on the CPDP ability, all 20 projects will be used as target domain data in turn during our empirical study.
- The CPDP model building process consists of two intertwined parts, *i.e.,* transfer learning and defect prediction model building.
  - Transfer learning aims to augment data from different domains by selecting relevant instances or assigning appropriate weights to different instances, *etc.* Table 1 outlines the parameters of the transfer learning techniques considered in our empirical study.
  - Based on the adapted data, many off-the-shelf classification techniques can be applied to build the defect prediction model. Table 2 outlines the parameters of the classification techniques considered in this paper.
- The performance of the defect prediction ability of the CPDP model is at the end evaluated upon the hold-out set from the target domain data.

Note that there are 13 CPDP techniques considered in our empirical study. All of them are either recognized as the state-of-the-art in the CPDP community or used as the baseline for many other follow-up CPDP techniques. Table 3 lists the combination of transfer learning and classifier used in each CPDP technique. To carry out the automated parameter optimization for a CPDP technique, `Hyperopt` is allocated 1,000 function evaluations. In our context, one function evaluation represents the complete model training process of a CPDP technique with a trial parameter setup, which can be computationally expensive. To carry out statistical analysis over our experiments, the optimization over each CPDP technique is repeated 10 times.

2.4 Performance Metric

To evaluate the performance of different CPDP methods for identifying defect-prone modules, we choose the area under the receiver operator characteristic curve (AUC) in our empirical study. This is because AUC is the most widely used performance metric in the defect prediction literature. In addition, there are two distinctive characteristics of AUC: 1) different from other prevalent metrics like precision and recall, the calculation of AUC does not depend on any threshold [75] which is difficult to tweak in order to carry out an unbiased assessment; and 2) it is not sensitive to imbalanced data which is not unusual in defect prediction [36]. Note that the

Table 2: The parameters of studied classifiers

	Parameters		
Classifiers	**Name**	**Description**	**Range**
K-Nearest Neighbor (KNN)	n_neighbors	The number of neighbors to each point (default=1) [N]	[1, 50]
Boost	n_estimators	The maximum number of estimators (default=50) [N]	[10, 1000]
	learning rate	A factor that shrinks the contribution of each classifier (default=1) [R]	[0.01, 10]
Classification and Regression Tree (CART)	criterion	The maximum number of estimators (default=10) [N]	[10, 100]
	max_features	The function to measure the quality of a split (default='gini') [C]	{'gini', 'entropy'}
	splitter	The number of features to consider when looking for the best split (default='auto') [C]	{'auto', 'sqrt', 'log2'}
	min_samples_split	The minimum number of samples required to split an internal node (default=2) [N]	[2, N_source/10]
Random Forest (RF)	n_estimators	The maximum number of estimators (default=10) [N]	[10, 100]
	criterion	The function to measure the quality of a split (default='gini') [C]	{'gini', 'entropy'}
	max_features	The number of features to consider when looking for the best split (default='auto') [C]	{'auto', 'sqrt', 'log2'}
	min_samples_split	The minimum number of samples required to split an internal node (default=2) [N]	[2, N_source/10]
Support Vector Machine (SVM)	kernel	The type of kernel (default='poly') [C]	{'rbf', 'linear', 'poly', 'sigmoid'}
	degree	Degree of the polynomial kernel function (default=3) [N]	[1, 5]
	coef0	Independent term in kernel function. It is only significant in 'poly' and 'sigmoid' (default=0.0) [R]	[0, 10]
	gamma	Kernel coefficient for 'rbf', 'poly' and 'sigmoid' (default=1) [R]	[1e-2, 1e2]
Multi-layer Perceptron (MLP)	active	Activation function for the hidden layer (default='relu') [C]	{'identity', 'logistic', 'tanh', 'relu'}
	alpha	L2 penalty (regularization term) parameter (default=0.0001) [R]	[1e-6, 1]
	iter	Maximum number of iterations (default=200) [N]	[10, 500]
Ridge	alpha	Regularization strength (default=1) [R]	[0.0001, 1000]
	normalize	Whether to standardize (default='False') [C]	{'True', 'False'}
Naive Bayes (NB)	NBType	The type of prior distribution (default='Gaussian') [C]	{'gaussian', 'multinomial', 'bernoulli'}

[N] An integer value from range
[R] A real value from range
[C] A choice from categories

Table 3: Overview of existing CPDP techniques considered in our empirical study.

CPDP Techniques	Reference	CPDP Techniques	Reference
Bruakfilter (NB)	[67]	DS+BF (RF)	[3]
Petersfilter (RF)	[56]	DS+BF (NB)	
Petersfilter (NB)		DTB	[11]
Petersfilter (KNN)		DBSCANfilter (RF)	[32]
FSS+Bagging (RF)	[22]	DBSCANfilter (NB)	
FSS+Bagging (NB)		DBSCANfilter (KNN)	
UM	[73]		

The classifier is shown in the brackets while outside part is the transfer learning technique. UM uses Universal to carry out transfer learning and Naive Bayes as a classifier. DTB uses DTB to carry out transfer learning part and Naive Bayes to conduct classification.

larger the AUC value, the better prediction accuracy of the underlying classification technique is. In particular, the AUC value ranges from 0 to 1 where 0 indicates the worst performance, 0.5 corresponds a random guessing performance and 1 represents the best performance.

3 RESULTS AND ANALYSIS

In this section, we will present the experimental results of our empirical study and address the research questions posed in Section 1.

3.1 On the Impacts of Automated Parameter Optimization Over CPDP Techniques

3.1.1 Research Method. To address **RQ1**, we investigate the magnitude of AUC performance improvement achieved by the CPDP model optimized by `Hyperopt` versus the one trained by its default parameter setting. Instead of comparing the difference of vanilla AUC values, we use Cohen's d effect size [13] to quantify such magnitude. Given two sets of samples say S_1 and S_2, Cohen's d effect size aims to provide a statistical measure of the standardized mean difference between them:

$$d = \frac{\mu_1 - \mu_2}{s}, \tag{1}$$

where μ_1 and μ_2 is the mean of S_1 and S_2 respectively; s is as defined as the pooled standard deviation:

$$s = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}} \tag{2}$$

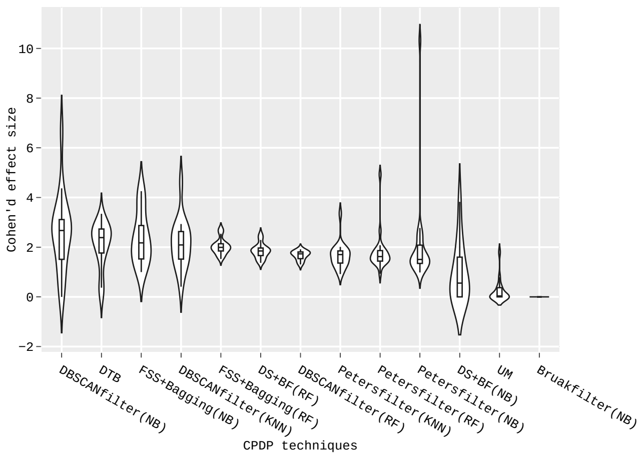

Figure 2: The AUC performance improvement in terms of Cohen's d effect size for each studied CPDP technique.

where n_1 and n_2 is the number of samples in S_1 and S_2 respectively; while s_1 and s_2 are the corresponding standard deviations of the two sample sets. To have a conceptual assessment of the magnitude, according to the suggestions in [61], $d < 0.2$ is defined as *negligible*, $0.2 < d \leq 0.5$ is treated as *small*, $0.5 < d \leq 0.8$ is regarded as *medium*, $0.8 < d \leq 1.0$ is large while it is *huge* if d goes beyond 1.0.

As introduced in Section 1, we run four different optimization types (as presented in Section 1) in parallel upon each baseline CPDP technique. To investigate whether `Hyperopt` can improve the performance of a CPDP technique, we only present the results from the best optimization type to make our conclusion sharper. For each CPDP technique, we use a violin plot to show the distributions of its median values of Cohen's d effect size obtained by optimizing the parameters of this CPDP techniques on 20 projects. To have a better understanding of the effect of automated parameter optimization upon different CPDP techniques, the violin plots are sorted, from left to right, by the median values of Cohen's d effect size in a descending order.

3.1.2 Results. From the comparison results shown in Figure 2, we clearly see that the performance of 12 out of 13 (around 92%) existing CPDP techniques have been improved after automated parameter optimization. In addition, according to the categorization in [61], the magnitudes of most AUC performance improvements are substantial and important. In particular, ten of them (around 77%) are classified as *huge*; while the performance improvements achieved by optimizing the parameters of DS+BF (NB) belong to the *medium* scale. In particular, `Hyperopt` leads to the most significant performance improvement on DBSCANfilter (NB) and DTB. On the other hand, the magnitudes of AUC performance improvement achieved by optimizing the parameters of UM and Bruakfilter (NB) are *negligible* (i.e., Cohen's $d < 0.2$). It is worth mentioning that `Hyperopt` cannot improve the performance of Bruakfilter (NB) any further at all considered projects. This suggests that the performance of Bruakfilter (NB) is not sensitive to its parameter setting. Overall, we obtain the following findings:

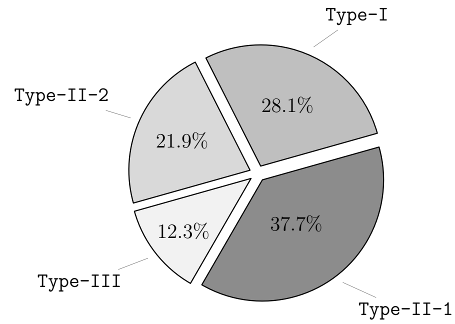

Figure 3: The percentage of significantly better performance achieved by different optimization types.

Answer to RQ1: ***Automated parameter optimization can improve the performance of defect prediction models in the context of CPDP. In particular, the performance of 10 out of 13 (around 77%) studied CPDP techniques have been improved substantially (i.e., huge in terms of Cohen's d effect size value).***

3.2 Comparing Different Types of Parameter Optimization

3.2.1 Research Method. To answer **RQ2**, we investigate the performance of four different types of parameter optimization, as introduced in Section 1. To have an overview of the result, for each optimization type, we record the number of times that its AUC value is significantly better than the other peers. In addition, to have a better intuition on the effect of different types of parameter optimization over each CPDP technique, we use violin plots to show the distributions of the median values of Cohen's d effect size obtained over 20 projects.

3.2.2 Results. From the pie chart shown in Figure 3, it is surprising to see that the type of only optimizing the parameters associated with the transfer learning part in CPDP is more likely to produce the best performance. In particular, 37.7% of the best AUC performance is achieved by `Type-II-1`. It is even better than simultaneously optimizing the parameters of both transfer learning and classification parts, i.e., `Type-I`, which wins on 28.1% comparisons. This might be because given the same amount of computational budget, simultaneously optimizing the parameters of both transfer learning and classification parts is very challenging due to the large search space. As a result, `Hyperopt` might run out of function evaluations before approaching the optimum on either part. On the other hand, if `Hyperopt` only focuses on optimizing the parameters of the transfer learning part, the search space is significantly reduced. Therefore, although only part of the parameters is considered, it is more likely to find the optimal parameter configuration of the transfer learning part within the limited number of function evaluations. The same applied to `Type-II-2`, which only focus on optimizing the parameters of the classification techniques. However, as shown in Figure 3, the performance of `Type-II-2` is not as competitive as `Type-I` and `Type-II-1`, implying that the classification part is less important than the transfer learning part in CPDP, which eventually obscures

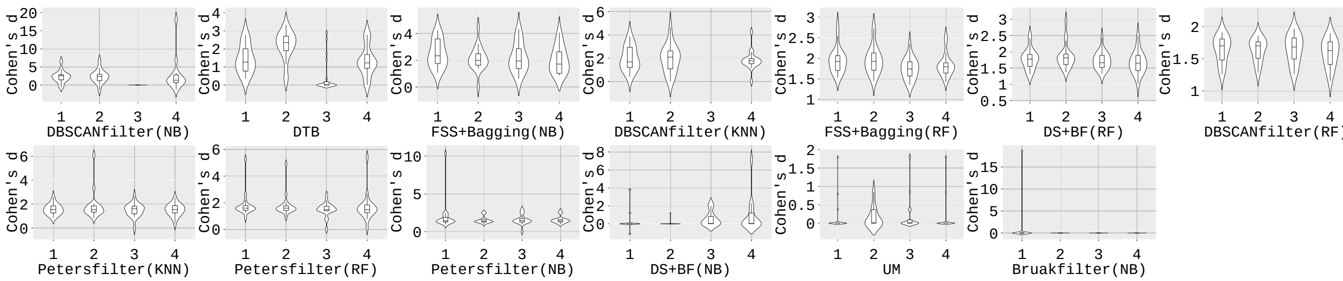

Figure 4: AUC performance improvement in terms of Cohen's d effect size for each studied CPDP technique. 1: `Type-I`, 2: `Type-II-1`, 3: `Type-II-2`, 4: `Type-III`

the benefit brought by the reduced search space. Finally, we see that sequentially optimizing the transfer learning and classification parts with equal budget of computation is the worst optimization type (`Type-III`). This is because it does not only fail to fully optimize both parts before exhausting the function evaluations, but also ignore the tentative coupling relationship between the parameters associated with both the transfer learning and classification.

From the results shown in Figure 4, we find that the performance difference between different types of parameter optimization is not very significant in most CPDP techniques. Nonetheless, we can still observe the superiority of `Type-I` and `Type-II-1` over the other two optimization types in most performance comparisons. In particular, for DBSCANfilter (NB), DBSCANfilter (KNN) and DTB, only optimizing the parameters of the classification part does not make any contribution to the performance improvement on the corresponding CPDP techniques. This observation is also aligned with our previous conclusion that the transfer learning part is more determinant than the classification part in CPDP. In summary, we find that:

> Answer to RQ2: ***Given a limited amount of computational budget, it is more plausible to focus on the parameter optimization of the transfer learning part in CPDP than the other types, including optimizing the configurations of both transfer learning and classification simultaneously. This observation also demonstrates that the transfer learning part is more determinant in CPDP.***

3.3 Comparing Different Combinations of Transfer Learning and Classification Techniques for CPDP

3.3.1 Research Method. To address **RQ3**, we build and compare 62 different CPDP models by combining those transfer learning and classification techniques listed in Table 1 and Table 2 respectively. 13 out of these 62 combinations exist in the literature. The remaining 49 combinations can be regarded as 'new' CPDP techniques. Because DTB requires to update the weights of its training instances during the training process, it cannot work with KNN or MLP which do not support online training data adjustments. In other words, the combinations DTB-KNN and DTB-MLP are ruled out from our empirical study. For clear observation, instead of presenting the performance of all 62 combinations together, we only show the 10 best CPDP techniques. We use violin plots to show the distributions of their AUC values.

In addition, for each project, we compare the performance of the best CPDP technique from the existing literature and those 'newly' developed in this paper. To have a statistically sound conclusion, we apply the Wilcoxon signed-rank sum test with a 0.05 significance level to validate the statistical significance of those comparisons. In particular, if the best candidate from the 'newly' developed CPDP techniques is significantly better than the one from the current literature, it is denoted as *win*; if their difference is not statistically significant, it is denoted as *tie*; otherwise, it is denoted as *loss*. We keep a record of the number of times of these three scenarios.

3.3.2 Results. From the violin plots shown in Figure 5, we find that the list of top 10 CPDP techniques varies from different projects. In particular, DTB-RF is the best CPDP technique as it appears in all top 10 lists and is ranked as the first place in 9 out of 20 projects. Furthermore, we notice that DTB, Peterfilter and DBSCANfilter are the best transfer learning techniques for CPDP because they were used as the transfer learning part in the CPDP techniques of all top 10 lists. From these observations, we conclude that CPDP techniques also follow the *no-free-lunch theorem* [70]. In other words, there is no universal CPDP technique capable of handling all CPDP tasks of the data have different characteristics.

Figure 6 gives the statistics of the comparison between the best CPDP technique from the existing literature and the one from our 'newly' developed portfolio. From the statistics, we can see that the CPDP techniques newly developed in this paper, by making a novel combination of transfer learning and classification techniques, are better than those existing ones in the literature. Remarkably, they are only outperformed by the existing CPDP techniques under one occasion. From this observation, we envisage that the current research on CPDP is far from mature. There is no rule-of-thumb a practitioner can follow to design a CPDP technique for the black-box dataset at hand. For **RQ3**, we have the following findings:

> Answer to RQ3: ***The current research on CPDP techniques is far from mature. Given a CPDP task, there is no rule-of-thumb available for a practitioner to follow in order to 1) design an effective technique that carries out an appropriate transfer learning and classification; and 2) find out the optimal configurations of their associated parameters.***

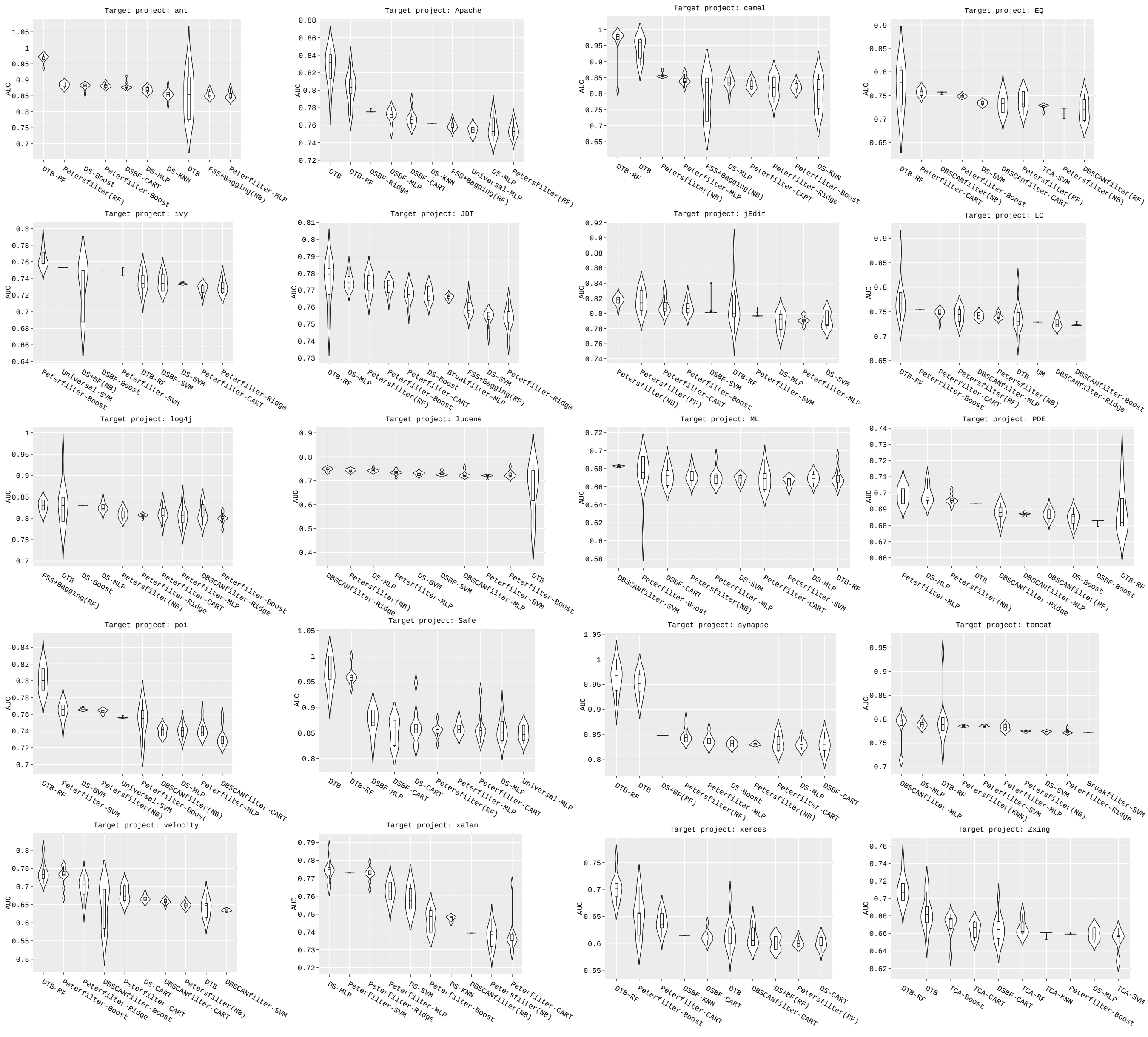

Figure 5: Violin plots AUC values obtained by top 10 CPDP techniques for different projects.

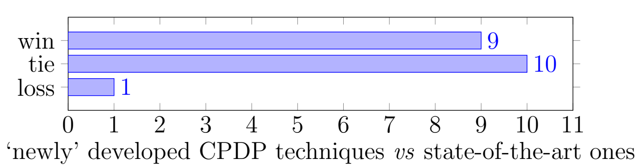

Figure 6: Comparisons of state-of-the-art CPDP techniques and those 'newly' developed in this paper.

4 RELATED WORK

Software Defect Prediction (SDP) is one of the most active research areas in software engineering [16], [33],[43], [49], [57], [76]. A defect predictor can find the most defect-prone software components based on the historical project data, which offers great help for the software engineers to allocate limited resources on those defect-prone components at testing or maintenance stages. To this end, certain sets of metrics are usually extracted from the historical data in order to build a predictive model. These metrics can be categorized into two types: code metrics and process metrics. Code metrics represent complexity, such as Size [2], McCabe [39], CK [12], OO

[17]. The principle behind code metrics is that more complex code tends to cause defects. As for process metrics, the more changes made in code, the more defect can be caused. Some popular process metrics include Relative code change churn [47], Change [46], Change Entropy [20], Micro interaction metrics [34]. In addition to code and process metrics, researches on metrics based on domain knowledge exist, such as network measure [42] and anti-pattern [63]. In this case, Li et al. [36] provide a latest analysis on SDP.

The aim of defect prediction is to learn an accurate model (supervised or unsupervised) from a corpus of data (e.g., static code features, churn data, defect information) and apply the model to new and unseen data. The training data can be from the same project, i.e., within project defect prediction (WPDP) or (the majority or the entirety) from other projects, i.e., cross-project defect prediction (CPDP). For supervised methods, CPDP is composed of two parts: domain adaptation and classification. For each component, a specific algorithm is applied to achieve the function of the component, i.e., a transfer technique for domain adaptation and a classifier for classification. In the past two decades, CPDP has attracted an increasing attention, as evidenced by many survey work, such as, [27], [25], [28], [75]. In CPDP, the homogeneous CPDP problem, which we focus in this work, refers to the case where the metrics of the source and target projects are the same or have the same ones. In contrast, when the metrics of the source and target projects are different, then it is denoted as heterogeneous CPDP problem. In 2009, Zimmermann et al. [77] made the first feasibility analysis about CPDP. Later on, He et al. [23] also made an investigation on the feasibility of CPDP. Their reviews have discussed different methods to handle CPDP: instance selection is the earliest and most common method in CPDP, in which similar source instances to the target instances are selected to learn a model, including instance filtering [67], [56], [32], [60], removing outlier [3]. Alternatively, instance weighting uses different weights for each source instance, depending on the relevance of the instance to the target instances [38], [11], [59]. Researches has been conducted on data selection to select proper projects to learn a model when there are multiple source projects, such as, [22], [23], [29], [24]. The results have revealed that not all instance features can benefit CPDP. Therefore, they proposed to remove useless features in training, which is also known as feature selection [21]. Instance standardization is another type of methods to handle CPDP, for which the key idea is to transform source and target instance into a similar form (e.g: distribution, representation) [50], [73].

Researches exist that aim to improve the performance of CPDP by using different models [10], model combination [72], [52], [26] or a model ensemble [68], [74]. Existing studies focus not only on the prediction performance of CPDP but also on the efficiency [58], [9] and privacy [53], [54], [55]. Another solution to improve prediction performance is via data preprocessing. To date, many preprocessing methods have been used in various CPDP algorithms [69], [14], [66], and unsupervised methods have also been used, such as, [48].

Given the above various techniques proposed for SDP, it is challenge to make an appropriate selection. What make it even more difficult is that, most of these techniques involve more or less some parameters that are crucial to the performance of the algorithm. As a result, parameter optimization is essential to achieve the best performance. However, there is very little work discussing parameter selection for SDP. Among others, Lessmann et al. [35] optimized the parameters by using grid search. Tantithamthavorn et al. [64] performed the first attempt to optimizing algorithm parameters automatically on SDP and investigated the impact of automated parameter optimization on defect prediction models [65]. Recently, Agrawal et al. [1] studied how to optimize the parameters of analysis tool "DODGE".

Unlike others, our work is, to the best of our knowledge, the first comprehensive empirical study about the impact of automated parameter optimization on transfer learning for CPDP, based on which we have revealed important findings and insights that have not been known before.

5 DISCUSSIONS

5.1 Insights Learned from Our Empirical Study

Our empirical study, for the first time, reveals some important and interesting facts that provide new insights to further advance the research on CPDP.

Through our comprehensive experiments, we have shown that automated parameter optimization can significantly improve the defect prediction performance of various CPDP techniques with a *huge* effect size in general. In particular, it is surprising but also exciting to realize that optimizing the parameters of transfer learning part only is generally the most cost effective way of tuning the defect prediction performance of CPDP techniques. Such a finding offers important insight and guidance for future research: given a limited amount of computational budget, designing sophisticated optimizer for the parameters of CPDP techniques can start off by solely considering the transfer learning part, without compromising the performance.

Our other insightful finding is to reveal the fact that the current research on CPDP is far from mature. In particular, many state-of-the-art combinations of transfer learning and classification techniques are flawed, and that the best combination can be case dependent. This suggests that automatically finding the optimal combination of CPDP techniques for a case is a vital research direction, and more importantly, the combination should be tuned with respect to the optimization of parameters. Such an observation can derive a brand new direction of research, namely the portfolio optimization of transfer learning and classifier that lead to an automated design of CPDP technique.

5.2 Threats to Validity

Similar to many empirical studies in software engineering, our work is subject to threats to validity. Specifically, internal threats can be related to the number of function evaluations used in the optimization. Indeed, a larger amount of function evaluations may lead to better results in some cases. However, the function evaluation in our empirical study is expensive and time consuming, as every single one needs to go through the full machine learning training process, validation and testing. As a result, a proper choice should be a good trade-off between the performance and time. To mitigate this threat, we have run numbers of options in a trial-and-error manner. We then concluded that 1,000 function evaluations is deemed as a balanced choice without compromising the validity of our conclusions. Further, to mitigate bias, we repeated 10 times for each CPDP technique on a project, which is acceptable considering the cost of function evaluation.

Construct threats can be raised from the selected quality indicator. In this work, AUC have been chosen as the key performance indicator in our empirical comparisons. The selection is mainly driven by its simplicity (no extra parameter is required) and its robustness (insensitive to imbalanced data). In addition, AUC have been widely recognised as one of the most reliable performance indicator in the machine learning community [37].

Finally, external threats are concerned with the dataset and CPDP techniques studied. To mitigate such, we have studied 62 CPDP techniques, including 13 combinations from existing work on CPDP and 49 other combinations that are new to the CPDP community but widely applied in classic machine learning research. Further, as discussed in Section 2.1, our studied dataset covers a wide spectrum of the real-world defected projects with diverse nature, each of which was selected based on six systematic criteria. Such tailored setting, although not exhaustive, is not uncommon in empirical software engineering and can serve as strong foundation to generalize our findings, especially considering that an exhaustive study of all possible CPDP techniques and dataset is unrealistic.

6 CONCLUSIONS

In the paper, we conduct the first empirical study, which offers an in-depth understanding on the impacts of automated parameter optimization for CPDP based on 62 CPDP techniques and 20 real-world projects. Our results reveal that:

- Automated parameter optimization can significantly improve the CPDP techniques. Up to 77% of the improvement exhibits *huge* effect size under the Cohen's rule.
- Optimizing the parameters of transfer learning techniques plays a more important role in performance improvement in CPDP.
- The state-of-the-arts combinations of transfer learning and classification are far from mature, as the statistically best technique comes from the 49 new combinations in most cases.

Our findings provide valuable insights for the practitioners from this particular research field to consider. Drawing on such, in our future work, we will design sophisticated optimizer for CPDP that explicitly searches the parameter space for the transfer learning part. Furthermore, the problem of portfolio optimization for CPDP, which involves both the selection of combination and parameter tuning, is also one of our ongoing research directions.

REFERENCES

[1] Amritanshu Agrawal, Wei Fu, Di Chen, Xipeng Shen, and Tim Menzies. 2019. How to "DODGE" Complex Software Analytics? *CoRR* abs/1902.01838 (2019).

[2] Fumio Akiyama. 1971. An Example of Software System Debugging. In *IFIP Congress (1)*. 353–359.

[3] Sousuke Amasaki, Kazuya Kawata, and Tomoyuki Yokogawa. 2015. Improving Cross-Project Defect Prediction Methods with Data Simplification. In *41st Euromicro Conference on Software Engineering and Advanced Applications, EUROMICRO-SEAA 2015, Madeira, Portugal, August 26-28, 2015*. 96–103.

[4] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *NIPS'11: Proc. of the 25th Annual Conference on Neural Information Processing Systems*. 2546–2554.

[5] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13 (2012), 281–305.

[6] James Bergstra, Daniel Yamins, and David D. Cox. 2013. Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms. In *SCIPY'13: Proc. of the 12th Python in Science Conference*. 13–20.

[7] James Bergstra, Daniel Yamins, and David D. Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *ICML'13: Proc. of the 30th International Conference on Machine Learning*. 115–123.

[8] Lionel C. Briand, Walcélio L. Melo, and Jürgen Wüst. 2002. Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects. *IEEE Trans. Software Eng.* 28, 7 (2002), 706–720.

[9] Gerardo Canfora, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. 2013. Multi-objective Cross-Project Defect Prediction. In *Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg, March 18-22, 2013*. 252–261.

[10] Qimeng Cao, Qing Sun, Qinghua Cao, and Huobin Tan. 2015. Software defect prediction via transfer learning based neural network. *2015 First International Conference on Reliability Systems Engineering (ICRSE)* (2015), 1–10.

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

[12] Shyam R. Chidamber and Chris F. Kemerer. 1994. A Metrics Suite for Object Oriented Design. *IEEE Trans. Software Eng.* 20, 6 (1994), 476–493.

[13] Jacob Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences*. Routledge.

[14] Ana Erika Camargo Cruz and Koichiro Ochimizu. 2009. Towards logistic regression models for predicting fault-prone code across software projects. In *Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement, ESEM 2009, October 15-16, 2009, Lake Buena Vista, Florida, USA*. 460–463.

[15] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2010. An extensive comparison of bug prediction approaches. In *Proceedings of the 7th International Working Conference on Mining Software Repositories, MSR 2010 (Co-located with ICSE), Cape Town, South Africa, May 2-3, 2010, Proceedings*. 31–41.

[16] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2010. An extensive comparison of bug prediction approaches. In *MSR'10: Proc. of the 7th International Working Conference on Mining Software Repositories (Co-located with ICSE)*. 31–41.

[17] Fernando Brito e Abreu and Rogério Carapuça. 1994. Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems and Software* 26, 1 (1994), 87–96.

[18] Matthias Feurer and Frank Hutter. 2019. Hyperparameter Optimization. In *Automated Machine Learning - Methods, Systems, Challenges*. 3–33.

[19] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. 2012. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Trans. Software Eng.* 38, 6 (2012), 1276–1304.

[20] Ahmed E. Hassan. 2009. Predicting faults using the complexity of code changes. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*. 78–88.

[21] Peng He, Bing Li, Xiao Liu, Jun Chen, and Yutao Ma. 2015. An empirical study on software defect prediction with a simplified metric set. *Information & Software Technology* 59 (2015), 170–190.

[22] Zhimin He, Fayola Peters, Tim Menzies, and Ye Yang. 2013. Learning from Open-Source Projects: An Empirical Study on Defect Prediction. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, Maryland, USA, October 10-11, 2013*. 45–54.

[23] Zhimin He, Fengdi Shu, Ye Yang, Mingshu Li, and Qing Wang. 2012. An investigation on the feasibility of cross-project defect prediction. *Autom. Softw. Eng.* 19, 2 (2012), 167–199.

[24] Steffen Herbold. 2013. Training data selection for cross-project defect prediction. In *9th International Conference on Predictive Models in Software Engineering, PROMISE '13, Baltimore, MD, USA, October 9, 2013*. 6:1–6:10.

[25] Steffen Herbold. 2017. A systematic mapping study on cross-project defect prediction. *CoRR* abs/1705.06429 (2017).

[26] Steffen Herbold, Alexander Trautsch, and Jens Grabowski. 2017. Global vs. local models for cross-project defect prediction - A replication study. *Empirical Software Engineering* 22, 4 (2017), 1866–1902.

[27] Steffen Herbold, Alexander Trautsch, and Jens Grabowski. 2018. A Comparative Study to Benchmark Cross-Project Defect Prediction Approaches. *IEEE Trans. Software Eng.* 44, 9 (2018), 811–833.

[28] Seyedrebvar Hosseini, Burak Turhan, and Dimuthu Gunarathna. 2019. A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction. *IEEE Trans. Software Eng.* 45, 2 (2019), 111–147.

[29] Seyedrebvar Hosseini, Burak Turhan, and Mika Mäntylä. 2016. Search Based Training Data Selection For Cross Project Defect Prediction. In *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2016, Ciudad Real, Spain, September 9, 2016*.

3:1–3:10.

[30] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). 2019. *Automated Machine Learning - Methods, Systems, Challenges*. Springer.

[31] Marian Jureczko and Lech Madeyski. 2010. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering, PROMISE 2010, Timisoara, Romania, September 12-13, 2010*. 9.

[32] Kazuya Kawata, Sousuke Amasaki, and Tomoyuki Yokogawa. 2015. Improving Relevancy Filter Methods for Cross-Project Defect Prediction. In *2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence*. 2–7.

[33] Taek Lee, Jaechang Nam, DongGyun Han, Sunghun Kim, and Hoh Peter In. 2011. Micro interaction metrics for defect prediction. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*. 311–321.

[34] Taek Lee, Jaechang Nam, DongGyun Han, Sunghun Kim, and Hoh Peter In. 2011. Micro interaction metrics for defect prediction. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*. 311–321.

[35] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. 2008. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Trans. Software Eng.* 34, 4 (2008), 485–496.

[36] Zhiqiang Li, Xiao-Yuan Jing, and Xiaoke Zhu. 2018. Progress on approaches to software defect prediction. *IET Software* 12, 3 (2018), 161–175.

[37] Charles X. Ling, Jin Huang, and Harry Zhang. 2003. AUC: a Statistically Consistent and more Discriminating Measure than Accuracy. In *IJCAI'03: Proc. of the 8th International Joint Conference on Artificial Intelligence*. 519–526.

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

[39] Thomas J. McCabe. 1976. A Complexity Measure. *IEEE Trans. Software Eng.* 2, 4 (1976), 308–320.

[40] Thilo Mende. 2010. Replication of defect prediction studies: problems, pitfalls and recommendations. In *PROMISE'10: Proc. of the 6th International Conference on Predictive Models in Software Engineering*. 5.

[41] Thilo Mende and Rainer Koschke. 2009. Revisiting the evaluation of defect prediction models. In *PROMISE'09: Proc. of the 5th International Workshop on Predictive Models in Software Engineering*. 7.

[42] Andrew Meneely, Laurie Williams, Will Snipes, and Jason A. Osborne. 2008. Predicting failures with developer networks and social network analysis. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, Atlanta, Georgia, USA, November 9-14, 2008*. 13–23.

[43] Tim Menzies, Jeremy Greenwald, and Art Frank. 2007. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Trans. Software Eng.* 33, 1 (2007), 2–13.

[44] Tim Menzies and Martin J. Shepperd. 2012. Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering* 17, 1-2 (2012), 1–17.

[45] Nikolaos Mittas and Lefteris Angelis. 2013. Ranking and Clustering Software Cost Estimation Models through a Multiple Comparisons Algorithm. *IEEE Trans. Software Eng.* 39, 4 (2013), 537–551.

[46] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. 2008. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*. 181–190.

[47] Nachiappan Nagappan and Thomas Ball. 2005. Use of relative code churn measures to predict system defect density. In *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*. 284–292.

[48] Jaechang Nam and Sunghun Kim. 2015. CLAMI: Defect Prediction on Unlabeled Datasets (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*. 452–463.

[49] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. 2013. Transfer defect learning. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*. 382–391.

[50] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. 2013. Transfer defect learning. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*. 382–391.

[51] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Trans. Knowl. Data Eng.* 22, 10 (2010), 1345–1359.

[52] Annibale Panichella, Rocco Oliveto, and Andrea De Lucia. 2014. Cross-project defect prediction models: L'Union fait la force. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014*. 164–173.

[53] Fayola Peters and Tim Menzies. 2012. Privacy and utility for defect prediction: Experiments with MORPH. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. 189–199.

[54] Fayola Peters, Tim Menzies, Liang Gong, and Hongyu Zhang. 2013. Balancing Privacy and Utility in Cross-Company Defect Prediction. *IEEE Trans. Software Eng.* 39, 8 (2013), 1054–1068.

[55] Fayola Peters, Tim Menzies, and Lucas Layman. 2015. LACE2: Better Privacy-Preserving Data Sharing for Cross Project Defect Prediction. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. 801–811.

[56] Fayola Peters, Tim Menzies, and Andrian Marcus. 2013. Better cross company defect prediction. In *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*. 409–418.

[57] Foyzur Rahman, Daryl Posnett, Abram Hindle, Earl T. Barr, and Premkumar T. Devanbu. 2011. BugCache for inspections: hit or miss?. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*. 322–331.

[58] Duksan Ryu and Jongmoon Baik. 2016. Effective multi-objective naïve Bayes learning for cross-project defect prediction. *Appl. Soft Comput.* 49 (2016), 1062–1077.

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

[60] Duksan Ryu, J.-I Jang, and Jongmoon Baik. 2015. A Hybrid Instance Selection Using Nearest-Neighbor for Cross-Project Defect Prediction. *Journal of Computer Science and Technology* 30 (09 2015), 969–980.

[61] Shlomo Sawilowsky. 2009. New effect size rules of thumb. *Journal of Modern Applied Statistical Methods* 8 (2009), 467–474. Issue 2.

[62] Martin J. Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. 2013. Data Quality: Some Comments on the NASA Software Defect Datasets. *IEEE Trans. Software Eng.* 39, 9 (2013), 1208–1215.

[63] Seyyed Ehsan Salamati Taba, Foutse Khomh, Ying Zou, Ahmed E. Hassan, and Meiyappan Nagappan. 2013. Predicting Bugs Using Antipatterns. In *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013*. 270–279.

[64] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*. 321–332.

[65] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2019. The Impact of Automated Parameter Optimization on Defect Prediction Models. *IEEE Transactions on Software Engineering* 45, 7 (2019), 683–711.

[66] Haonan Tong, Bin Liu, Shihai Wang, and Qiuying Li. 2019. Transfer-Learning Oriented Class Imbalance Learning for Cross-Project Defect Prediction. *CoRR* abs/1901.08429 (2019).

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

[68] Satoshi Uchigaki, Shinji Uchida, Koji Toda, and Akito Monden. 2012. An Ensemble Approach of Simple Regression Models to Cross-Project Fault Prediction. In *13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2012, Kyoto, Japan, August 8-10, 2012*. 476–481.

[69] Shinya Watanabe, Haruhiko Kaiya, and Kenji Kaijiri. 2008. Adapting a Fault Prediction Model to Allow Inter Languagereuse. In *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering (PROMISE '08)*. 19–24.

[70] David H. Wolpert and William G. Macready. 1997. No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation* 1, 1 (1997), 67–82.

[71] Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. 2011. ReLink: recovering links between bugs and changes. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*. 15–25.

[72] Xin Xia, David Lo, Sinno Jialin Pan, Nachiappan Nagappan, and Xinyu Wang. 2016. HYDRA: Massively Compositional Model for Cross-Project Defect Prediction. *IEEE Trans. Software Eng.* 42, 10 (2016), 977–998.

[73] Feng Zhang, Audris Mockus, Iman Keivanloo, and Ying Zou. 2016. Towards building a universal defect prediction model with rank transformed predictors. *Empirical Software Engineering* 21, 5 (2016), 2107–2145.

[74] Yun Zhang, David Lo, Xin Xia, and Jianling Sun. 2015. An Empirical Study of Classifier Combination for Cross-Project Defect Prediction. In *39th IEEE Annual Computer Software and Applications Conference, COMPSAC 2015, Taichung, Taiwan, July 1-5, 2015. Volume 2*. 264–269.

[75] Yuming Zhou, Yibiao Yang, Hongmin Lu, Lin Chen, Yanhui Li, Yangyang Zhao, Junyan Qian, and Baowen Xu. 2018. How Far We Have Progressed in the Journey? An Examination of Cross-Project Defect Prediction. *ACM Trans. Softw. Eng. Methodol.* 27, 1 (2018), 1:1–1:51.

[76] Thomas Zimmermann and Nachiappan Nagappan. 2008. Predicting defects using network analysis on dependency graphs. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*. 531–540.

[77] Thomas Zimmermann, Nachiappan Nagappan, Harald C. Gall, Emanuel Giger, and Brendan Murphy. 2009. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *ESEC/FSE'09: Proc. of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 91–100.