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

Ke Li§, Zilin Xiang§, Tao Chen§, Shuo Wang¶, Kay Chen Tan∗

§College of Computer Science and Engineering, UESTC, Chengdu, 611731, China
¶Department of Computer Science, University of Exeter, Exeter, EX4 4QF, UK
∗Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong SAR

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/knowledge 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 parameter optimization. However, there is a lack of thorough understanding about what are the impacts of automated parameter 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

1 INTRODUCTION
According to the latest Annual Software Fail Watch report from Tricentis1, 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/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 [18]. 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 [48], cross-project defect prediction (CPDP) [8] has become increasingly popular as an effective way to deal with the shortage of training data [68]. 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 and adaptable parameters (87% prevalent classification techniques are with at least one parameter [57, 58]), the settings of which largely

influence the prediction accuracy when encountering unseen software projects [38, 39]. 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 [18] and be adverse to the research reproducibility of defect prediction studies [42, 43]. Recently, Tantithamthavorn et al. [57, 58] 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 becomes 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 [29], 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 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 can be used to adapt and 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 [48]. 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:

**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.

\(^2\)The parameters of a classification model is set as default values when optimizing the transfer learning part.

\(^3\)[http://hyperopt.github.io/hyperopt/]
- 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:

- 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 threats to validity are discussed in Section 4 along with a pragmatic literature review 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 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 overfitting 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 [24, 26, 27, 68], we shortlist five publicly available datasets (i.e., JURECZKO, NASA, SOFTLAB, AEEEM, RelLink).

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 [55] 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 [14]: 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 churn.

- RelLink [64]: 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 [30]: 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\textsuperscript{2} 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.

\textsuperscript{2}http://hyperopt.github.io/hyperopt/
- 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 [17]. 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.

2.3 Architecture for Optimizing CPDP Model

Figure 1 shows the architecture of using Hyperopt to optimize the performance of CPDP models. As shown in Figure 1, the implementation steps of our empirical study are given below.

(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 is used for training whilst the data from the target domain is used for testing. 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.

(2) 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.

(3) 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 [68] 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 [34]. Note that the larger the AUC values, 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. Under a target domain (project), instead of comparing the difference of vanilla AUC values\(^6\) for all repeated runs, we use Cohen’s \(d\) effect size\(^1\) to quantify such magnitude. This is because it is simple to calculate and has been predominately used as the metric for automated parameter optimization of defect prediction model\(^{[57]}\). 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:

\[
d = \frac{\mu_1 - \mu_2}{s}
\]

where \(\mu_1\) and \(\mu_2\) is the mean of \(S_1\) and \(S_2\) respectively; \(s\) is as defined as the pooled standard deviation:

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

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\(^{[54]}\), \(d < 0.2\) 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 technique 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\(^{[54]}\), 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

---

\(^{[6]}\)To make our results self-contained, the vanilla AUC values are given in the supplementary document of this paper and can be found in https://github.com/COLA-Laboratory/icse2020

---

<table>
<thead>
<tr>
<th>Transfer learning</th>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bruakfilter</td>
<td>k</td>
<td>The number of neighbors to each point (default=10)(^{[N]})</td>
<td>([1, 100])</td>
<td></td>
</tr>
<tr>
<td>DS</td>
<td>topN</td>
<td>The number of closest training sets (default=5)(^{[N]})</td>
<td>([1, 15])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FSS</td>
<td>The ratio of unstable features filtered out (default=0.2)(^{[R]})</td>
<td>([0.1, 0.5])</td>
<td></td>
</tr>
<tr>
<td>DSBF</td>
<td>Toprank</td>
<td>The number assigned to 1 when performing feature reduction (default=1)(^{[N]})</td>
<td>([1, 10])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>k</td>
<td>The number of neighbors to each point (default=25)(^{[N]})</td>
<td>([1, 100])</td>
<td></td>
</tr>
<tr>
<td>TCA</td>
<td>kernel</td>
<td>The type of kernel (default=’linear’)(^{[C]})</td>
<td>{'primal’, ‘linear’, ‘rbf’, ‘sam’}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dimension</td>
<td>The dimension after transforming (default=5)(^{[N]})</td>
<td>([5, \max(N\text{-}source, N\text{-}target)])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lamb</td>
<td>Lambda value in equation (default=1)(^{[R]})</td>
<td>([0.000001, 100])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gamma</td>
<td>Kernel bandwidth for ‘rbf’ kernel (default=1)(^{[R]})</td>
<td>([0.00001, 100])</td>
<td></td>
</tr>
<tr>
<td>DBSCANfilter</td>
<td>eps</td>
<td>The maximum distance between two samples for one to be considered as in the neighborhood of the other (default=1.0)(^{[R]})</td>
<td>([0.1, 100])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_samples</td>
<td>The number of samples (or total weight) in a neighborhood for a point to be considered as a core point (default=10)(^{[N]})</td>
<td>([1, 100])</td>
<td></td>
</tr>
<tr>
<td>Universal</td>
<td>p-value</td>
<td>The associated p-value for statistical test (default=0.05)(^{[R]})</td>
<td>([0.01, 0.1])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>quantifyType</td>
<td>The type of quantifying the difference between distributions (default=’cliff’)(^{[C]})</td>
<td>{'cliff’, ‘cohen’}</td>
<td></td>
</tr>
<tr>
<td>DTB</td>
<td>k</td>
<td>The number of neighbors to each point (default=10)(^{[N]})</td>
<td>([1, 50])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>The maximum number of iterations (default=20)(^{[N]})</td>
<td>([5, 30])</td>
<td></td>
</tr>
<tr>
<td>Peterfilter</td>
<td>r</td>
<td>The number of points in each cluster (default=10)(^{[N]})</td>
<td>([1, 100])</td>
<td></td>
</tr>
</tbody>
</table>

\(^{[N]}\) An integer value from range
\(^{[R]}\) A real value from range
\(^{[C]}\) A choice from categories

---

Table 1: Parameters of the transfer learning techniques considered in our experiments
Table 2: Parameters of the classification techniques considered in our experiments

<table>
<thead>
<tr>
<th>Classification techniques</th>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-Nearest Neighbor (KNN)</td>
<td>n_neighbors</td>
<td>The number of neighbors to each point (default=1)</td>
<td>[1, 50]</td>
<td></td>
</tr>
<tr>
<td>Boost</td>
<td>n_estimators</td>
<td>The maximum number of estimators (default=50)</td>
<td>[10, 1000]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>learning_rate</td>
<td>A factor that shrinks the contribution of each classifier (default=1)</td>
<td>[0.01, 10]</td>
<td></td>
</tr>
<tr>
<td>Classification and Regression Tree (CART)</td>
<td>criterion</td>
<td>The maximum number of estimators (default=10)</td>
<td>[10, 100]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_features</td>
<td>The function to measure the quality of a split (default='gini')</td>
<td>{'gini', 'entropy'}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>splitter</td>
<td>The number of features to consider when looking for the best split (default='auto')</td>
<td>{'auto', 'sqrt', 'log2'}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_samples_split</td>
<td>The minimum number of samples required to split an internal node (default=2)</td>
<td>[2, N_source/10]</td>
<td></td>
</tr>
<tr>
<td>Random Forest (RF)</td>
<td>n_estimators</td>
<td>The maximum number of estimators (default=10)</td>
<td>[10, 100]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>criterion</td>
<td>The function to measure the quality of a split (default='gini')</td>
<td>{'gini', 'entropy'}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_features</td>
<td>The number of features to consider when looking for the best split (default='auto')</td>
<td>{'auto', 'sqrt', 'log2'}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_samples_split</td>
<td>The minimum number of samples required to split an internal node (default=2)</td>
<td>[2, N_source/10]</td>
<td></td>
</tr>
<tr>
<td>Support Vector Machine (SVM)</td>
<td>kernel</td>
<td>The type of kernel (default= poly')</td>
<td>{'rbf', 'linear', 'poly', 'sigmoid'}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>degree</td>
<td>Degree of the polynomial kernel function (default=3)</td>
<td>[1, 5]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>coef0</td>
<td>Independent term in kernel function. It is only significant in 'poly' and 'sigmoid' (default=0)</td>
<td>[0, 10]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gamma</td>
<td>Kernel coefficient for 'rbf', 'poly' and 'sigmoid' (default=1)</td>
<td>[0.01, 100]</td>
<td></td>
</tr>
<tr>
<td>Multi-layer Perceptron (MLP)</td>
<td>active</td>
<td>Activation function for the hidden layer (default= relu)</td>
<td>{'identity', 'logistic', 'tanh', 'relu'}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>L2 penalty (regularization term) parameter (default=0.0001)</td>
<td>[0.000001, 1]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iter</td>
<td>Maximum number of iterations (default=200)</td>
<td>[10, 500]</td>
<td></td>
</tr>
<tr>
<td>Ridge</td>
<td>alpha</td>
<td>Regularization strength (default=1)</td>
<td>[0.001, 1000]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>normalize</td>
<td>Whether to standardize (default=False)</td>
<td>{'True', 'False'}</td>
<td></td>
</tr>
<tr>
<td>Naive Bayes (NB)</td>
<td>NBType</td>
<td>The type of prior distribution (default= Gaussian)</td>
<td>{'gaussian', 'multinomial', 'bernoulli'}</td>
<td></td>
</tr>
</tbody>
</table>

[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.

<table>
<thead>
<tr>
<th>CPDP Techniques</th>
<th>Reference</th>
<th>CPDP Techniques</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bruakfilter (NB)</td>
<td>[60]</td>
<td>ISS+BF (RF)</td>
<td>[3]</td>
</tr>
<tr>
<td>Petersfilter (RF)</td>
<td>[50]</td>
<td>ISS+BF (NB)</td>
<td></td>
</tr>
<tr>
<td>Petersfilter (NB)</td>
<td></td>
<td>DTR</td>
<td>[10]</td>
</tr>
<tr>
<td>Petersfilter (KNN)</td>
<td></td>
<td>DBSCANfilter (RF)</td>
<td></td>
</tr>
<tr>
<td>FSS+Bagging (RF)</td>
<td>[21]</td>
<td>DBSCANfilter (NB)</td>
<td>[31]</td>
</tr>
<tr>
<td>FSS+Bagging (NB)</td>
<td></td>
<td>DBSCANfilter (KNN)</td>
<td></td>
</tr>
<tr>
<td>UM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 DB to carry out transfer learning part and Naive Bayes to conduct classification.

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 might suggest that the original parameter configuration of Bruakfilter (NB) is already optimal. Overall, we obtain the following findings:

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 CPDP technique, we record the best parameter optimization type. In addition, for each optimization type, we also record the number of times that its AUC value is significantly better than the other peers over all CPDP technique and project pairs. 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 results shown in Figure 3, we have a general impression that Type-II-1 plays as the best parameter optimization type in most cases. In particular, for UM and DTB, Type-II-1 almost dominates the list. The second best parameter optimization type is Type-I whilst the worst one is Type-III which is rarely ranked as the best optimization type in most cases.

The pie chart shown in Figure 4 is a more integrated way to summerize the results collected from Figure 3. From this figure, we can see that the type of only optimizing the parameters associated with the transfer learning part in CPDP is indeed 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 neither 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 4, 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 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 5, 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 a better visualization, 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 best one from the current literature, it is denoted as \( \text{win} \); if their difference is not statistically significant, it is denoted as \( \text{tie} \); otherwise, it is denoted as \( \text{loss} \). We keep a record of the number of times of these three scenarios.

3.3.2 Results. From the violin plots shown in Figure 6, 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, Petersfilter 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 [63]. In other words, there is no universal CPDP technique capable of handling all CPDP tasks of the data have different characteristics.

Figure 7 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 for a practitioner to 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.

4 DISCUSSIONS

4.1 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.
4.2 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
we empirically study the auto-
Our findings, apart from confirm-
The number of combinations
work, refers to the case where the metrics of the source and tar-
which was selected based on six systematic criteria. Such a tailored
learned a model when there are multiple source projects [21, 22],
[52]. Projects and feature selection is another way to transfer the
common way to transfer the learned model for CPDP, in which
consists of two parts: domain adaptation and classification where
the former is resolved by transfer learning while the latter is tackled
by a machine learning classifier.
In the past two decades, CPDP has attracted an increasing at-
ments constitutes 62 CPDP techniques. This amount is
nearly six times more than the 11 classifiers studied by Tan-
ethal. [57]. However, our work is fundamentally
different from theirs in the following aspects:

- **Considering cross-projects:** We empirically study the auto-
parameter optimization for transfer learning based
models on cross-project prediction while Tantithamthavorn
et al. [57] focus on the optimization for the defect prediction
model within a single project.

- **Studying a larger set of models:** The number of combinations
of transfer learner and classifier considered in our exper-
iments constitutes 62 CPDP techniques. This amount is
nearly six times more than the 11 classifiers studied by Tan-
ethal. [57].

- **Providing wider insights:** Our findings, apart from confirm-
ing that the automated parameter optimization on CPDP
is effective, also provides insights on the algorithm selec-
tions for CPDP. In contrast, Tantithamthavorn et al. [57]
mainly provide analysis on the effectiveness and stability of
automated parameter optimization.

In summary, 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.

6 CONCLUSIONS

In this 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 across 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 ‘newly’ developed 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.

ACKNOWLEDGEMENT

K. Li was supported by UKRI Future Leaders Fellowship (Grant No. MR/S017062/1) and Royal Society (Grant No. IEC/NSFC/170243).

REFERENCES


[34] Zhiqiang Li, Xiao-Yuan Jing, and Xiaoke Zhu. 2018. Progress on approaches to software defect prediction: IET Software 12, 3 (2018), 163–175.


