Heterogeneous Defect Prediction

Jaechang Nam, Wei Fu, *Student Member, IEEE*, Sunghun Kim, *Member, IEEE*, Tim Menzies, *Member, IEEE*, and Lin Tan, *Member, IEEE*

Abstract—Many recent studies have documented the success of cross-project defect prediction (CPDP) to predict defects for new projects lacking in defect data by using prediction models built by other projects. However, most studies share the same limitations: it requires homogeneous data; i.e., different projects must describe themselves using the *same* metrics. This paper presents methods for *heterogeneous* defect prediction (HDP) that matches up different metrics in different projects. Metric matching for HDP requires a "large enough" sample of distributions in the source and target projects—which raises the question on how large is "large enough" for effective heterogeneous defect prediction. This paper shows that empirically and theoretically, "large enough" may be very small indeed. For example, using a mathematical model of defect prediction, we identify categories of data sets were as few as 50 instances are enough to build a defect prediction model. Our conclusion for this work is that, even when projects use different metric sets, it is possible to quickly transfer lessons learned about defect prediction.

Index Terms—Defect prediction, quality assurance, heterogeneous metrics, transfer learning

1 INTRODUCTION

MACHINE learners can be used to automatically generate software quality models from project data [1], [2]. Such data comprises various *software metrics* and *labels*:

- *Software metrics* are the terms used to describe software projects. Commonly used software metrics for defect prediction are complexity metrics (such as lines of code, Halstead metrics, McCabe's cyclometic complexity, and CK metrics) and process metrics [3], [4], [5], [6].
- When learning defect models, *labels* indicate whether the source code is buggy or clean for binary classification [7], [8].

Most proposed defect prediction models have been evaluated on "within-project" defect prediction (WPDP) settings [1], [2], [7]. As shown in Fig. 1a, in WPDP, each instance representing a source code file or function consists of software metric values and is labeled as buggy or clean. In the WPDP setting, a prediction model is trained using the labeled instances in *Project A* and predict unlabeled ('?') instances in the same project as buggy or clean.

Sometimes, software engineers need more than within-project defect prediction. The 21st century is the era of the "mash up", where new systems are built by combining large sections of old code in some new and novel manner. Software engineers working on such mash-ups often face the problem of working with large code bases built by other developers that are, in some sense "alien"; i.e., code has been written for other purposes, by other people, for different organizations. When performing quality assurance on such code, developers seek some way to "transfer" whatever expertise is available and apply it to the "alien" code. Specifically, for this paper, we assume that

- Developers are experts on their local code base;
- Developers have applied that expertise to log what parts of their code are particularly defect-prone;
- Developers now want to apply that defect log to build defect predictors for the "alien" code.

Prior papers have explored transferring data about code quality from one project across to another. For example, researchers have proposed "cross-project" defect prediction (CPDP) [8], [9], [10], [11], [12], [13]. CPDP approaches predict defects even for new projects lacking in historical data by reusing information from other projects. As shown in Fig. 1b, in CPDP, a prediction model is trained by labeled instances in *Project A* (source) and predicts defects in *Project B* (target).

Most CPDP approaches have a serious limitation: typical CPDP requires that all projects collect exactly the same metrics (as shown in Fig. 1b). Developers deal with this limitation by collecting the same metric sets. However, there are several situations where collecting the same metric sets can be challenging. Language-dependent metrics are difficult to collect for projects written in different languages. Metrics collected by a commercial metric tool with a limited license may generate additional cost for project teams when collecting metrics for new projects that do not obtain the tool license. Because of these situations, publicly available defect

- *J. Nam is with the School of Computer Science and Electrical Engineering, Handong Global University, Pohang, Korea. E-mail: jcnam@handong.edu.*
- *W. Fu and T. Menzies are with the Department of Computer Science, North Carolina State University, Raleigh, NC 27695. E-mail: wfu@ncsu.edu, tim.menzies@gmail.com.*
- *S. Kim is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China. E-mail: hunkim@cse.ust.hk.*
- *L. Tan is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. E-mail: lintan@uwaterloo.ca.*

Manuscript received 19 Jan. 2016; revised 31 Mar. 2017; accepted 13 June 2017. Date of publication 0 . 0000; date of current version 0 . 0000.
(Corresponding author: Jaechang Nam.)
Recommended for acceptance by A. Hasan.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2017.2720603

0098-5589 © 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

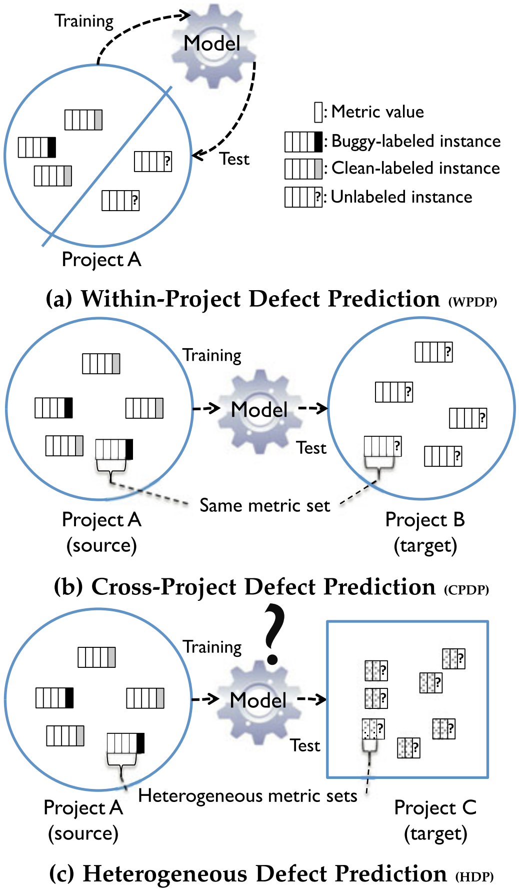


Fig. 1. Various defect prediction scenarios.

datasets that are widely used in defect prediction literature usually have *heterogeneous* metric sets:

- In heterogeneous data, different metrics are collected in different projects.
- For example, many NASA datasets in the PROMISE repository have 37 metrics but AEEEM datasets used by D'Ambros et al. have 61 metrics [1], [14]. The only common metric between NASA and AEEEM datasets is *lines of code (LOC)*. CPDP between NASA and AEEEM datasets with all metric sets is not feasible since they have completely different metrics [12].

Some CPDP studies use only common metrics when source and target datasets have heterogeneous metric sets [10], [12]. For example, Turhan et al. use the only 17 common metrics between the NASA and SOFTLAB datasets that have heterogeneous metric sets [12]. This approach is hardly a general solution since finding other projects with multiple common metrics can be challenging. As mentioned, there is only one common metric between NASA and AEEEM. Also, only using common metrics may degrade the performance of CPDP models. That is because some informative metrics necessary for building a good prediction model may not be in the common metrics across datasets. For example, the CPDP approach proposed by Turhan et al. did not outperform WPDP in terms of the average f-measure (0.35 versus 0.39) [12].

In this paper, we propose the heterogeneous defect prediction (HDP) approach to predict defects across projects even with heterogeneous metric sets. If the proposed approach is feasible as in Fig. 1c, we could reuse any existing defect datasets to build a prediction model. For example, many PROMISE defect datasets even if they have heterogeneous metric sets [14] could be used as training datasets to predict defects in any project. Thus, addressing the issue of the heterogeneous metric sets also can benefit developers who want to build a prediction model with more defects from publicly available defect datasets even whose source code is not available.

The key idea of our HDP approach is to transfer knowledge, i.e., the typical defect-proneness tendency of software metrics, from a source dataset to predict defects in a target dataset by matching metrics that have similar distributions between source and target datasets [1], [2], [6], [15], [16]. In addition, we also used metric selection to remove less informative metrics of a source dataset for a prediction model before metric matching.

In addition to proposing HDP, it is important to identify the lower bounds of the sizes of the source and target datasets for effective transfer learning since HDP compares distributions between source and target datasets. If HDP requires many source or target instances to compare there distributions, HDP may not be effective and efficient to build a prediction model. We address this limit experimentally as well as theoretically in this paper.

1.1 Research Questions

To systematically evaluate HDP models, we set two research questions.

- RQ1: Is heterogeneous defect prediction comparable to WPDP, existing CPDP approaches for heterogeneous metric sets, and unsupervised defect prediction?
- RQ2: What are the lower bounds of the size of source and target datasets for effective HDP?

1.2 Contributions

Our experimental results on RQ1 (in Section 6) show that HDP models are feasible and their prediction performance is promising. About 47.2-83.1 percent of HDP predictions are better or comparable to predictions in baseline approaches with statistical significance.

A natural response to the RQ1 results is to ask RQ2; i.e., how early is such transfer feasible? Section 7 shows some curious empirical results that show a few hundred examples are enough—this result is curious since we would have thought that heterogeneous transfer would complicate move information across projects; thus *increasing* the quantity of data needed for effective transfer.

The results of Section 7 are so curious that is natural to ask: are they just a quirk of our data, or do they represent a more general case? To answer this question and to assess the external validity of the results in Section 7, Section 8 of this paper builds and explores a mathematical model of defect prediction. That analysis concludes that Section 7 is actually representative of the general case; i.e., transfer should be possible after a mere few hundred examples.

Our contributions are summarized as follows:

- Proposing the heterogeneous defect prediction models.
- Conducting extensive and large-scale experiments to evaluate the heterogeneous defect prediction models.

- Empirically validating the lower bounds of the size of source and target datasets for effective heterogeneous defect prediction.
- Theoretically demonstrating that the above empirical results are actually the general and expected results.

1.3 Extensions from Prior Publication

We extend the previous conference paper of the same name [17] in the following ways. First, we motivate this study in the view of transfer learning in software engineering (SE). Thus, we discuss how transfer learning can be helpful to understand the nature of generality in SE and why we focus on defect prediction in terms of transfer learning (Section 2). Second, we address new research question about the effective sizes of source and target datasets when conducting HDP. In Sections 7 and 8, we show experimental and theoretical validation to investigate the effective sizes of project datasets for HDP. Third, we discuss more related work with recent studies. In Section 3.2, we discuss metric sets used in CPDP and how our HDP is similar to and different from recent studies about CPDP using heterogeneous metric sets.

2 Motivation

2.1 Why Explore Transfer Learning?

One reason to explore transfer learning is to study the nature of generality in SE. Professional societies assume such generalities exist when they offer lists of supposedly general "best practices":

- For example, the IEEE 1012 standard for software verification [18] proposes numerous methods for assessing software quality;
- Endres and Rombach catalog dozens of lessons of software engineering [19] such as McCabe's Law (functions with a "cyclomatic complexity" greater than ten are more error prone);
- Further, many other widely-cited researchers do the same such as Jones [20] and Glass [21] who list (for exmple) Brooks' Law (adding programmers to a late project makes it later).
- More generally, Budgen and Kitchenham seek to reorganize SE research using general conclusions drawn from a larger number of studies [22], [23].

Given the constant pace of change within SE, can we trust those supposed generalities? Numerous *local learning* results show that we should mistrust general conclusions (made over a wide population of projects) since they may not hold for projects [24], [25]. Posnett et al. [26] discuss *ecological inference* in software engineering, which is the concept that what holds for the entire population also holds for each individual. They learn models at different levels of aggregation (modules, packages, and files) and show that models working at one level of aggregation can be sub-optimal at others. For example, Yang et al. [27], Bettenburg et al. [25], and Menzies et al. [24] all explore the generation of models using *all* data versus *local* samples that are more specific to particular test cases. These papers report that better models (sometimes with much lower variance in their predictions) are generated from local information. These results have an unsettling effect on anyone struggling to propose policies for an organization. If all prior conclusions can change for the new project, or some small part of a project, how can any manager ever hope to propose and defend IT policies (e.g., when should some module be inspected, when should it be refactored, where to focus expensive testing procedures, etc.)?

If we cannot *generalize* to all projects and all parts of current projects, perhaps a more achievable goal is to *stabilize* the pace of conclusion change. While it may be a fool's errand and wait for eternal and global SE conclusions, one possible approach is for organizations to declare N prior projects as *reference projects*, from which lessons learned will be transferred to new projects. In practice, using such reference sets requires three processes:

1) Finding the reference sets (this paper shows that finding them may not require extensive and protracted data collection, at least for defect prediction).
2) Recognizing when to update the reference set. In practice, this could be as simple as noting when predictions start failing for new projects—at which time, we would loop to the point 1).
3) Transferring lessons from the reference set to new projects.

In the case where all the datasets use the same metrics, this is a relatively simple task. Krishna et al. [28] have found such reference projects just by training of a project X then testing on a project Y (and the reference set are the project Xs with highest scores). Once found, these reference sets can generate policies of an organization that are stable just as long as the reference set is not updated.

In this paper, we do not address the pace of change in the reference set (that is left for future work). Rather, we focus on the point 3): transferring lessons from the reference set to new projects in the case of heterogeneous data sets. To support this third point, we need to resolve the problem that this paper addresses, i.e., data expressed in different terminology cannot transfer till there is enough data to match old projects to new projects.

2.2 Why Explore Defect Prediction?

There are many lessons we *might* try to transfer between projects about staffing policies, testing methods, language choices, etc. While all those matters are important and are worthy of research, this section discusses why we focus on defect prediction.

Human programmers are clever, but flawed. Coding adds functionality, but also defects. Hence, software sometimes crashes (perhaps at the most awkward or dangerous moment) or delivers the wrong functionality. For a very long list of software-related errors, see Peter Neumann's "Risk Digest" at http://catless.ncl.ac.uk/Risks.

Since programming inherently introduces defects into programs, it is important to test them before they're used. Testing is expensive. Software assessment budgets are finite while assessment effectiveness increases exponentially with assessment effort. For example, for black-box testing methods, a *linear* increase in the confidence C of finding defects can take *exponentially* more effort.[1] Exponential costs

1. A randomly selected input to a program will find a fault with probability p. After N random black-box tests, the chances of the inputs not revealing any fault is $(1-p)^N$. Hence, the chances C of seeing the fault is $1-(1-p)^N$ which can be rearranged to $N(C,p) = log(1-C)/log(1-p)$. For example, $N(0.90, 10^{-3}) = 2301$ but $N(0.98, 10^{-3}) = 3901$; i.e., nearly double the number of tests.

quickly exhaust finite resources so standard practice is to apply the best available methods on code sections that seem most critical. But any method that focuses on parts of the code can blind us to defects in other areas. Some *lightweight sampling policy* should be used to explore the rest of the system. This sampling policy will always be incomplete. Nevertheless, it is the only option when resources prevent a complete assessment of everything.

One such lightweight sampling policy is defect predictors learned from software metrics such as static code attributes. For example, given static code descriptors for each module, plus a count of the number of issues raised during inspect (or at runtime), data miners can learn where the probability of software defects is highest.

The rest of this section argues that such defect predictors are *easy to use*, *widely-used*, and *useful* to use.

Easy to Use: Various software metrics such as static code attributes and process metrics can be automatically collected, even for very large systems, from software repositories [3], [4], [5], [6], [29]. Other methods, like manual code reviews, are far slower and far more labor-intensive. For example, depending on the review methods, 8 to 20 LOC/minute can be inspected and this effort repeats for all members of the review team, which can be as large as four or six people [30].

Widely Used: Researchers and industrial practitioners use the software metrics to guide software quality predictions. Defect prediction models have been reported at large industrial companies such as Google [31], Microsoft [32], AT&T [33], and Samsung [34]. Verification and validation (V&V) textbooks [35] advise using the software metrics to decide which modules are worth manual inspections.

Useful: Defect predictors often find the location of 70 percent (or more) of the defects in code [36]. Defect predictors have some level of generality: predictors learned at NASA [36] have also been found useful elsewhere (e.g., in Turkey [37], [38]). The success of this method in predictors in finding bugs is markedly higher than other currently-used industrial methods such as manual code reviews. For example, a panel at *IEEE Metrics 2002* [39] concluded that manual software reviews can find $\approx$60 percent of defects. In another work, Raffo documents the typical defect detection capability of industrial review methods: around 50 percent for full Fagan inspections [40] to 21 percent for less-structured inspections. In some sense, defect prediction might not be necessary for small software projects. However, software projects seldom grow by small fractions in practice. For example, a project team may suddenly merge a large branch into a master branch in a version control system or add a large third-part library. In addition, a small project could be just one of many other projects in a software company. In this case, the small project also should be considered for limited resource allocation in terms of software quality control by the company. For this reason, defect prediction could be useful even for the small software projects in practice.

Not only do defect predictors perform well compared to manual methods, they also are competitive with certain automatic methods. A recent study at ICSE'14, Rahman et al. [41] compared (a) static code analysis tools FindBugs, Jlint, and Pmd and (b) defect predictors (which they called "statistical defect prediction") built using logistic regression. They found no significant differences in the cost-effectiveness of these approaches. Given this equivalence, it is significant to note that defect prediction can be quickly adapted to new languages by building lightweight parsers to extract high-level software metrics. The same is not true for static code analyzers—these need extensive modification before they can be used on new languages.

Having offered general high-level notes on defect prediction, the next section describes in detail the related work on this topic.

3 Related Work

3.1 Related Work on Transfer Learning

In the machine learning literature, the 2010 article by Pan and Yang [42] is the definitive definition of transfer learning.

Pan and Yang state that transfer learning is defined over a *domain* D, which is composed of pairs of examples X and a probability distribution about those examples $P(X)$; i.e., $D = \{X, P(X)\}$. This P distribution represents what class values to expect, given the X values.

The transfer learning *task* T is to learn a function f that predicts labels Y; i.e., $T = \{Y, f\}$. Given a new example x, the intent is that the function can produce a correct label $y \in Y$; i.e., $y = f(x)$ and $x \in X$. According to Pan and Yang, synonyms for transfer learning include, learning to learn, life-long learning, knowledge transfer, inductive transfer, multitask learning, knowledge consolidation, context-sensitive learning, knowledge-based inductive bias, metalearning, and incremental/cumulative learning.

Pan and Yang [42] define four types of transfer learning:

- When moving from some source domain to the target domain, *instance-transfer* methods provide example data for model building in the target;
- *Feature-representation transfer* synthesizes example data for model building;
- *Parameter transfer* provides parameter terms for existing models;
- and *Relational-transfer* provides mappings between term parameters.

From a business perspective, we can offer the following examples of how to use these four kinds of transfer. Take the case where a company is moving from Java-based desktop application development to Python-based web application development. The project manager for the first Python webapp wants to build a model that helps her predict which classes have the most defects so that she can focus on system testing:

- *Instance-transfer* tells her which Java project data are relevant for building her Python defect prediction model.
- *Feature-representation transfer* will create synthesized Python project data based on analysis of the Java project data that she can use to build her defect prediction model.
- If defect prediction models previously existed for the Java projects, *parameter transfer* will tell her how to weight the terms in old models to make those model are relevant for the Python projects.

- Finally, *relational-transfer* will tell her how to translate some JAVA-specific concepts (such as metrics collected from JAVA interfaces classes) into synonymous terms for Python (note that this last kind of transfer is very difficult and, in the case of SE, the least explored).

In the SE literature, methods for CPDP using same/common metrics sets are examples of *instance transfer*. As to the other kinds of transfer, there is some work in the effort estimation literature of using genetic algorithms to automatically learn weights for different parameters [43]. Such work is an example of *parameter transfer*. To the best of our knowledge, there is no work on feature-representation transfer, but research into automatically learning APIs between programs [44] might be considered a close analog.

In the survey of Pan and Yang [42], most transfer learning algorithms in these four types of transfer learning assume the same feature space. In other words, the surveyed transfer learning studies in [42] focused on different distributions between source and target 'domains or tasks' under the assumption that the feature spaces between source and target domains are same. However, Pan and Yang discussed the need for transfer learning between source and target that have different feature spaces and referred to this kind of transfer learning as *heterogeneous transfer learning* [42].

A recent survey of transfer learning by Weiss et al. published in 2016 [45] categorizes transfer learning approaches in homogeneous or heterogeneous transfer learning based on the same or different feature spaces respectively. Weiss et al. put the four types of transfer learning by Pan and Yang into homogeneous transfer learning [45]. For heterogeneous transfer learning, Weiss et al. divide related studies into two sub-categories: *symmetric transformation* and *asymmetric transformation* [45]. Symmetric transformation finds a common latent space whether both source and target can have similar distributions while Asymmetric transformation aligns source and target features to form the same feature spaces [45].

By the definition of Weiss et al., HDP is an example of heterogeneous transfer learning based on asymmetric transformation to solve issues of CPDP using heterogeneous metric sets. We discuss the related work about CPDP based on transfer learning concept in the following section.

3.2 Related Work on Defect Prediction

Recall from the above that we distinguish cross-project defect prediction (CPDP) from within-project defect prediction (WPDP). The CPDP approaches have been studied by many researchers of late [8], [10], [11], [12], [13], [46], [47], [48], [49], [50]. Since the performance of CPDP is usually very poor [13], researchers have proposed various techniques to improve CPDP [8], [10], [12], [46], [47], [48], [49], [51]. In this section, we discuss CPDP studies in terms of metric sets in defect prediction datasets.

3.2.1 CPDP Using Same/Common Metric Sets

Watanabe et al. proposed the metric compensation approach for CPDP [51]. The metric compensation transforms a target dataset similar to a source dataset by using the average metric values [51]. To evaluate the performance of the metric compensation, Watanabe et al. collected two defect datasets with the same metric set (8 object-oriented metrics) from two software projects and then conducted CPDP [51].

Rahman et al. evaluated the CPDP performance in terms of cost-effectiveness and confirmed that the prediction performance of CPDP is comparable to WPDP [11]. For the empirical study, Rahman et al. collected 9 datasets with the same process metric set [11].

Fukushima et al. conducted an empirical study of just-in-time defect prediction in the CPDP setting [52]. They used 16 datasets with the same metric set [52]. The 11 datasets were provided by Kamei et al. but 5 projects were newly collected with the same metric set used in the 11 datasets [52], [53].

However, collecting datasets with the same metric set might limit CPDP. For example, if existing defect datasets contain object-oriented metrics such as CK metrics [3], collecting the same object-oriented metrics is impossible for projects that are written in non-object-oriented languages.

Turhan et al. proposed the nearest-neighbour (NN) filter to improve the performance of CPDP [12]. The basic idea of the NN filter is that prediction models are built by source instances that are nearest-neighbours of target instances [12]. To conduct CPDP, Turhan et al. used 10 NASA and SOFTLAB datasets in the PROMISE repository [12], [14].

Ma et al. proposed Transfer Naive Bayes (TNB) [10]. The TNB builds a prediction model by weighting source instances similar to target instances [10]. Using the same datasets used by Turhan et al., Ma et al. evaluated the TNB models for CPDP [10], [12].

Since the datasets used in the empirical studies of Turhan et al. and Ma et al. have heterogeneous metric sets, they conducted CPDP using the common metrics [10], [12]. There is another CPDP study with the top-K common metric subset [54]. However, as explained in Section 1, CPDP using common metrics is worse than WPDP [12], [54].

Nam et al. adapted a state-of-the-art transfer learning technique called Transfer Component Analysis (TCA) and proposed TCA+ [8]. They used 8 datasets in two groups, ReLink and AEEEM, with 26 and 61 metrics respectively [8].

However, Nam et al. could not conduct CPDP between ReLink and AEEEM because they have heterogeneous metric sets. Since the project pool with the same metric set is very limited, conducting CPDP using a project group with the same metric set can be limited as well. For example, at most 18 percent of defect datasets in the PROMISE repository have the same metric set [14]. In other words, we cannot directly conduct CPDP for the 18 percent of the defect datasets by using the remaining (82 percent) datasets in the PROMISE repository [14].

There are other CPDP studies using datasets with the same metric sets or using common metric sets [14], [24], [46], [47], [48], [49], [50]. Menzies et al. proposed a local prediction model based on clustering [24]. They used seven defect datasets with 20 object-oriented metrics from the PROMISE repository [14], [24]. Canfora et al., Panichella et al., and Zhang et al. used 10 Java projects only with the same metric set from the PROMISE repository [14], [46], [47], [50]. Ryu et al. proposed the value-cognitive boosting and transfer cost-sensitive boosting approaches for CPDP [48], [49].

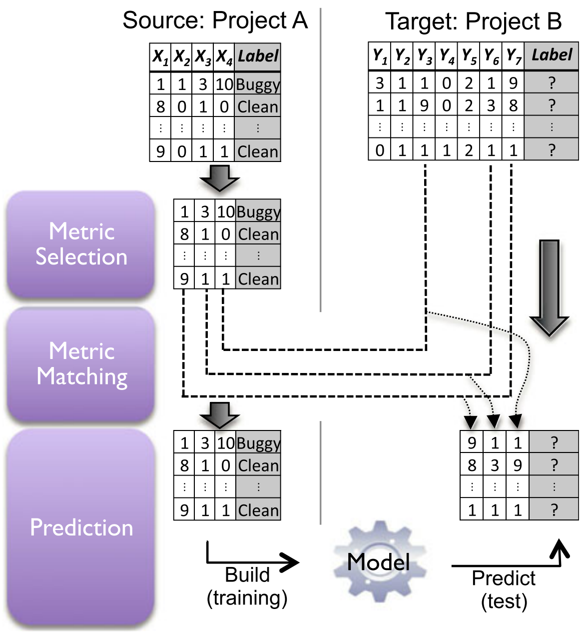


Fig. 2. Heterogeneous defect prediction.

Ryu et al. used common metrics in NASA and SOFTLAB datasets [48] or Jureczko datasets with the same metric set from the PROMISE repository [49]. These recent studies for CPDP did not discuss about the heterogeneity of metrics across project datasets.

Zhang et al. proposed the universal model for CPDP [55]. The universal model is built using 1,398 projects from SourceForge and Google code and leads to comparable prediction results to WPDP in their experimental setting [55].

However, the universal defect prediction model may be difficult to apply for the projects with heterogeneous metric sets since the universal model uses 26 metrics including code metrics, object-oriented metrics, and process metrics. In other words, the model can only be applicable for target datasets with the same 26 metrics. In the case where the target project has not been developed in object-oriented languages, a universal model built using object-oriented metrics cannot be used for the target dataset.

3.2.2 *CPDP Using Heterogeneous Metric Sets*

He et al. [56] addressed the limitations due to heterogeneous metric sets in CPDP studies listed above. Their approach, CPDP-IFS, used distribution characteristic vectors of an instance as metrics. The prediction performance of their best approach is comparable to or helpful in improving regular CPDP models [56].

However, the approach by He et al. is not compared with WPDP [56]. Although their best approach is helpful to improve regular CPDP models, the evaluation might be weak since the prediction performance of a regular CPDP is usually very poor [13]. In addition, He et al. conducted experiments on only 11 projects in 3 dataset groups [56].

Jing et al. proposed heterogeneous cross-company defect prediction based on the extended canonical correlation analysis (CCA+) [57] to address the limitations of heterogeneous metric sets. Their approach adds dummy metrics with zero values for non-existing metrics in source or target datasets and then transforms both source and target datasets to make their distributions similar. CCA+ was evaluated on 14 projects in four dataset groups.

We propose HDP to address the above limitations caused by projects with heterogeneous metric sets. Contrary to the study by He et al. [56], we compare HDP to WPDP, and HDP achieved better or comparable prediction performance to WPDP in about 71 percent of predictions. Comparing to the experiments for CCA+ [57] with 14 projects, we conducted more extensive experiments with 34 projects in 5 dataset groups. In addition, CCA+ transforms original source and target datasets so that it is difficult to directly explain the meaning of metric values generated by CCA+ [57]. However, HDP keeps the original metrics and builds models with the small subset of selected and matched metrics between source and target datasets in that it can make prediction models simpler and easier to explain [17], [58]. In Section 4, we describe our approach in detail.

4 APPROACH

Fig. 2 shows the overview of HDP based on metric selection and metric matching. In the figure, we have two datasets, Source and Target, with heterogeneous metric sets. Each row and column of a dataset represents an instance and a metric, respectively, and the last column represents instance labels. As shown in the figure, the metric sets in the source and target datasets are not identical (X_1 to X_4 and Y_1 to Y_7 respectively).

When given source and target datasets with heterogeneous metric sets, for metric selection we first apply a feature selection technique to the source. Feature selection is a common approach used in machine learning for selecting a subset of features by removing redundant and irrelevant features [59]. We apply widely used feature selection techniques for metric selection of a source dataset as in Section 4.1 [60], [61].

After that, metrics based on their similarity such as distribution or correlation between the source and target metrics are matched up. In Fig. 2, three target metrics are matched with the same number of source metrics.

After these processes, we finally arrive at a matched source and target metric set. With the final source dataset, HDP builds a model and predicts labels of target instances.

In the following sections, we explain the metric selection and matching in detail.

4.1 Metric Selection in Source Datasets

For metric selection, we used various feature selection approaches widely used in defect prediction such as gain ratio, chi-square, relief-F, and significance attribute evaluation [60], [61]. In our experiments, we used Weka implementation for these four feature selection approaches [62] According to benchmark studies about various feature selection approaches, a single best feature selection approach for all prediction models does not exist [63], [64], [65]. For this reason, we conduct experiments under different feature selection approaches. When applying feature selection approaches, we select top 15 percent of metrics as suggested by Gao et al. [60]. For example, if the number of features in a dataset is 200, we select 30 top features ranked by a feature selection approach. In addition, we compare

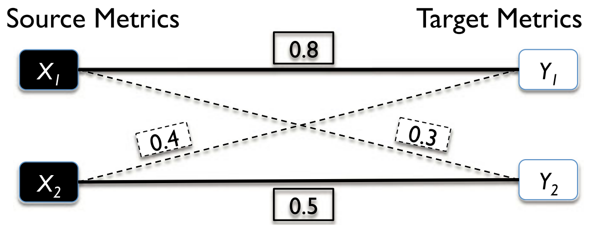


Fig. 3. An example of metric matching between source and target datasets.

the prediction results with or without metric selection in the experiments.

4.2 Matching Source and Target Metrics

Matching source and target metrics is the core of HDP. The intuition of matching metrics is originated from the typical defect-proneness tendency of software metrics, i.e., the higher complexity of source code and development process causes the more defect-proneness [1], [2], [66]. The higher complexity of source code and development process is usually represented with the higher metric values. Thus, various product and process metrics, e.g., McCabe's cyclomatic, lines of code, and the number of developers modifying a file, follow this defect-proneness tendency [1], [2], [6], [15], [16]. By matching metrics, HDP transfers this defect-proneness tendency from a source project for predicting defects in a target project. For example, assume that a metric, the number of methods invoked by a class (*RFC*), in a certain Java project (source) has the tendency that a class file having the RFC value greater than 40 is highly defect-prone. If a target metric, *the number of operands*, follows the similar distribution and its defect-proneness tendency, transferring this defect-proneness tendency of the source metric, RFC, as knowledge by matching the source and target metrics could be effective to predict defects in the target dataset.

To match source and target metrics, we measure the similarity of each source and target metric pair by using several existing methods such as percentiles, Kolmogorov-Smirnov Test, and Spearman's correlation coefficient [67], [68]. We define metric matching analyzers as follows:

- Percentile based matching (PAnalyzer)
- Kolmogorov-Smirnov Test based matching (KSAnalyzer)
- Spearman's correlation based matching (SCoAnalyzer)

The key idea of these analyzers is computing matching scores for all pairs between the source and target metrics. Fig. 3 shows a sample matching. There are two source metrics (X_1 and X_2) and two target metrics (Y_1 and Y_2). Thus, there are four possible matching pairs, (X_1, Y_1), (X_1, Y_2), (X_2, Y_1), and (X_2, Y_2). The numbers in rectangles between matched source and target metrics in Fig. 3 represent matching scores computed by an analyzer. For example, the matching score between the metrics, X_1 and Y_1, is 0.8.

From all pairs between the source and target metrics, we remove poorly matched metrics whose matching score is not greater than a specific cutoff threshold. For example, if the matching score cutoff threshold is 0.3, we include only the matched metrics whose matching score is greater than 0.3. In Fig. 3, the edge (X_1, Y_2) in matched metrics will be excluded when the cutoff threshold is 0.3. Thus, all the candidate matching pairs we can consider include the edges (X_1, Y_1), (X_2, Y_2), and (X_2, Y_1) in this example. In Section 5, we design our empirical study under different matching score cutoff thresholds to investigate their impact on prediction.

We may not have any matched metrics based on the cutoff threshold. In this case, we cannot conduct defect prediction. In Fig. 3, if the cutoff threshold is 0.9, none of the matched metrics are considered for HDP so we cannot build a prediction model for the target dataset. For this reason, we investigate target prediction coverage (i.e., what percentage of target datasets could be predicted?) in our experiments.

After applying the cutoff threshold, we used *the maximum weighted bipartite matching* [69] technique to select a group of matched metrics, whose sum of matching scores is highest, without duplicated metrics. In Fig. 3, after applying the cutoff threshold of 0.30, we can form two groups of matched metrics without duplicated metrics. The first group consists of the edges, (X_1, Y_1) and (X_2, Y_2), and another group consists of the edge (X_2, Y_1). In each group, there are no duplicated metrics. The sum of matching scores in the first group is 1.3 (=0.8+0.5) and that of the second group is 0.4. The first group has a greater sum (1.3) of matching scores than the second one (0.4). Thus, we select the first matching group as the set of matched metrics for the given source and target metrics with the cutoff threshold of 0.30 in this example.

Each analyzer for the metric matching scores is described in the following sections.

4.2.1 *PAnalyzer*

PAnalyzer simply compares nine percentiles (10th, 20th,..., 90th) of ordered values between source and target metrics. A percentile is a statistical measure that indicates the value at a specific percentage of observations in descriptive statistics. By comparing differences at the nine percentiles, we simulate the similarity between source and target metric values. The intuition of this analyzer comes from the assumption that the similar source and target metric values have similar statistical information. Since comparing only medians, i.e., 50th percentile just show one aspect of distributions of source and target metric values, we expand the comparison at the 9 spots of distributions of those metric values.

First, we compute the difference of nth percentiles in source and target metric values by the following equation:

$$P_{ij}(n) = \frac{sp_{ij}(n)}{bp_{ij}(n)}, \tag{1}$$

where $P_{ij}(n)$ is the comparison function for nth percentiles of ith source and jth target metrics, and $sp_{ij}(n)$ and $bp_{ij}(n)$ are smaller and bigger percentile values respectively at nth percentiles of ith source and jth target metrics. For example, if the 10th percentile of the source metric values is 20 and that of target metric values is 15, the difference is 0.75 ($P_{ij}(10) = 15/20 = 0.75$). Then, we repeat this calculation at the 20th, 30th,..., 90th percentiles.

Using this percentile comparison function, a matching score between source and target metrics is calculated by the following equation:

$$M_{ij} = \frac{\sum_{k=1}^{9} P_{ij}(10 \times k)}{9}, \quad (2)$$

where M_{ij} is a matching score between ith source and jth target metrics. For example, if we assume a set of all P_{ij}, i.e., $P_{ij}(10 \times k) = \{0.75, 0.34, 0.23, 0.44, 0.55, 0.56, 0.78, 0.97, 0.55\}$, M_{ij} will be 0.574(=$\frac{0.75+0.34+0.23+0.44+0.55+0.56+0.78+0.97+0.55}{9}$). The best matching score of this equation is 1.0 when the values of the source and target metrics of all 9 percentiles are the same, i.e., $P_{ij}(n) = 1$.

4.2.2 KSAnalyzer

KSAnalyzer uses a p-value from the Kolmogorov-Smirnov Test (KS-test) as a matching score between source and target metrics. The KS-test is a non-parametric statistical test to compare two samples [67], [70]. Particularly, the KS-test can be applicable when we cannot be sure about the normality of two samples and/or the same variance [67], [70]. Since metrics in some defect datasets used in our empirical study have exponential distributions [2] and metrics in other datasets have unknown distributions and variances, the KS-test is a suitable statistical test to compare two metrics.

In the KS-test, a p-value shows the significance level with which we have very strong evidence to reject the null hypothesis, i.e., two samples are drawn from the same distribution [67], [70]. We expected that matched metrics whose null hypothesis can be rejected with significance levels specified by commonly used p-values such as 0.01, 0.05, and 0.10 can be filtered out to build a better prediction model. Thus, we used a p-value of the KS-test to decide the matched metrics should be filtered out. We used the *KolmogorovSmirnovTest* implemented in the *Apache commons math3 3.3* library.

The matching score is

$$M_{ij} = p_{ij}, \quad (3)$$

where p_{ij} is a p-value from the KS-test of ith source and jth target metrics. Note that in KSAnalyzer the higher matching score does not represent the higher similarity of two metrics. To observe how the matching scores based on the KS-test impact on prediction performance, we conducted experiments with various p-values.

4.2.3 SCoAnalyzer

In SCoAnalyzer, we used the Spearman's rank correlation coefficient as a matching score for source and target metrics [68]. Spearman's rank correlation measures how two samples are correlated [68]. To compute the coefficient, we used the *SpearmansCorrelation* in the *Apache commons math3 3.3* library. Since the size of metric vectors should be the same to compute the coefficient, we randomly select metric values from a metric vector that is of a greater size than another metric vector. For example, if the sizes of the source and target metric vectors are 110 and 100 respectively, we randomly select 100 metric values from the source metric to agree to the size between the source and target metrics. All metric values are sorted before computing the coefficient.

The matching score is as follows:

$$M_{ij} = c_{ij}, \quad (4)$$

TABLE 1
The 34 Defect Datasets from Five Groups

Group	Dataset	# of instances		# of metrics	Prediction Granularity
		All	Buggy(%)		
AEEEM [1], [8]	EQ	324	129 (39.81%)	61	Class
	JDT	997	206 (20.66%)		
	LC	691	64 (9.26%)		
	ML	1862	245 (13.16%)		
	PDE	1492	209 (14.01%)		
ReLink [71]	Apache	194	98 (50.52%)	26	File
	Safe	56	22 (39.29%)		
	ZXing	399	118 (29.57%)		
MORPH [72]	ant-1.3	125	20 (16.00%)	20	Class
	arc	234	27 (11.54%)		
	camel-1.0	339	13 (3.83%)		
	poi-1.5	237	141 (59.49%)		
	redaktor	176	27 (15.34%)		
	skarbonka	45	9 (20.00%)		
	tomcat	858	77 (8.97%)		
	velocity-1.4	196	147 (75.00%)		
	xalan-2.4	723	110 (15.21%)		
	xerces-1.2	440	71 (16.14%)		
NASA [14], [73]	cm1	344	42 (12.21%)	37	Function
	mw1	264	27 (10.23%)		
	pc1	759	61 (8.04%)		
	pc3	1125	140 (12.44%)		
	pc4	1399	178 (12.72%)		
	jm1	9593	1759 (18.34%)	21	
	pc2	1585	16 (1.01%)	36	
	pc5	17001	503 (2.96%)	38	
	mc1	9277	68 (0.73%)	38	
	mc2	127	44 (34.65%)	39	
	kc3	200	36 (18.00%)	39	
SOFTLAB [12]	ar1	121	9 (7.44%)	29	Function
	ar3	63	8 (12.70%)		
	ar4	107	20 (18.69%)		
	ar5	36	8 (22.22%)		
	ar6	101	15 (14.85%)		

where c_{ij} is a Spearman's rank correlation coefficient between ith source and jth target metrics.

4.3 Building Prediction Models

After applying metric selection and matching, we can finally build a prediction model using a source dataset with selected and matched metrics. Then, as a regular defect prediction model, we can predict defects on a target dataset with the matched metrics.

5 Experimental Setup

This section presents the details of our experimental study such as benchmark datasets, experimental design, and evaluation measures.

5.1 Benchmark Datasets

We collected publicly available datasets from previous studies [1], [8], [12], [71], [72]. Table 1 lists all dataset groups used in our experiments. Each dataset group has a heterogeneous metric set as shown in the table. Prediction Granularity in the last column of the table means the prediction granularity of instances. Since we focus on the distribution or correlation of metric values when matching metrics, it is beneficial to be able to apply the HDP approach on datasets even in different granularity levels.

We used five groups with 34 defect datasets: AEEEM, ReLink, MORPH, NASA, and SOFTLAB.

AEEEM was used to benchmark different defect prediction models [1] and to evaluate CPDP techniques [8], [56]. Each AEEEM dataset consists of 61 metrics including object-oriented (OO) metrics, previous-defect metrics, entropy metrics of change and code, and churn-of-source-code metrics [1].

Datasets in ReLink were used by Wu et al. [71] to improve the defect prediction performance by increasing the quality of the defect data and have 26 code complexity metrics extracted by the Understand tool [74].

The MORPH group contains defect datasets of several open source projects used in the study about the dataset privacy issue for defect prediction [72]. The 20 metrics used in MORPH are McCabe's cyclomatic metrics, CK metrics, and other OO metrics [72].

NASA and SOFTLAB contain proprietary datasets from NASA and a Turkish software company, respectively [12]. We used 11 NASA datasets in the PROMISE repository [14], [73]. Some NASA datasets have different metric sets as shown in Table 1. We used cleaned NASA datasets (DS$'$ version) available from the PROMISE repository [14], [73]. For the SOFTLAB group, we used all SOFTLAB datasets in the PROMISE repository [14]. The metrics used in both NASA and SOFTLAB groups are Halstead and McCabe's cyclomatic metrics but NASA has additional complexity metrics such as *parameter count* and *percentage of comments* [14].

Predicting defects is conducted across different dataset groups. For example, we build a prediction model by Apache in ReLink and tested the model on velocity-1.4 in MORPH (Apache⇒velocity-1.4).[2] Since some NASA datasets do not have the same metric sets, we also conducted cross prediction between some NASA datasets that have different metric sets, e.g., (cm1⇒jm1).

We did not conduct defect prediction across projects where datasets have the same metric set since the focus of our study is on prediction across datasets with heterogeneous metric sets. In total, we have 962 possible prediction combinations from these 34 datasets. Since we select top 15 percent of metrics from a source dataset for metric selection as explained in Section 4.1, the number of selected metrics varies from 3 (MORPH) to 9 (AEEEM) [60]. For datasets, we did not apply any data preprocessing approach such as log transformation [2] and sampling techniques for class imbalance [75] since the study focus is on the heterogeneous issue on CPDP datasets.

5.2 Cutoff Thresholds for Matching Scores

To build HDP models, we apply various cutoff thresholds for matching scores to observe how prediction performance varies according to different cutoff values. Matched metrics by analyzers have their own matching scores as explained in Section 4. We apply different cutoff values (0.05 and 0.10, 0.20,..., 0.90) for the HDP models. If a matching score cutoff is 0.50, we remove matched metrics with the matching score $\leq$ 0.50 and build a prediction model with matched metrics with the score $>$ 0.50. The number of matched metrics varies by each prediction combination. For example, when using KSAnalyzer with the cutoff of 0.05, the number of matched metrics is four in cm1⇒ar5 while that is one in ar6⇒pc3. The average number of matched metrics also varies by analyzers and cutoff values; 4 (PAnalyzer), 2 (KSAnalyzer), and 5 (SCoAnalyzer) in the cutoff of 0.05 but 1 (PAnalyzer), 1 (KSAnalyzer), and 4 (SCoAnalyzer) in the cutoff of 0.90.

5.3 Baselines

We compare HDP to four baselines: WPDP (Baseline1), CPDP using common metrics between source and target datasets (Baseline2), CPDP-IFS (Baseline3), and Unsupervised defect prediction (Baseline4).

We first compare HDP to WPDP. Comparing HDP to WPDP will provide empirical evidence of whether our HDP models are applicable in practice. When conducting WPDP, we applied feature selection approached to remove redundant and irrelevant features as suggested by Gao et al. [60]. To fairly compare WPDP with HDP, we used the same feature selection techniques used for metric selection in HDP as explained in Section 4.1 [60], [61].

We conduct CPDP using only common metrics (CPDP-CM) between source and target datasets as in previous CPDP studies [10], [12], [56]. For example, AEEEM and MORPH have OO metrics as common metrics so we select them to build prediction models for datasets between AEEEM and MORPH. Since selecting common metrics has been adopted to address the limitation on heterogeneous metric sets in previous CPDP studies [10], [12], [56], we set CPDP-CM as a baseline to evaluate our HDP models. The number of common metrics varies across the dataset groups as ranged from 1 to 38. Between AEEEM and ReLink, only one common metric exists, *LOC* (ck_oo_numberOfLinesOfCode : CountLineCode). Some NASA datasets that have different metric sets, e.g., pc5 versus mc2, have 38 common metrics. On average, the number of common metrics in our datasets is about 12. We put all the common metrics between the five dataset groups in the online appendix: *https://lifove.github.io/hdp/#cm*.

We include CPDP-IFS proposed by He et al. as a baseline [56]. CPDP-IFS enables defect prediction on projects with heterogeneous metric sets (Imbalanced Feature Sets) by using the 16 distribution characteristics of values of each instance with all metrics. The 16 distribution characteristics are mode, median, mean, harmonic mean, minimum, maximum, range, variation ratio, first quartile, third quartile, interquartile range, variance, standard deviation, coefficient of variance, skewness, and kurtosis [56]. The 16 distribution characteristics are used as features to build a prediction model [56].

As Baseline4, we add unsupervised defect prediction (UDP). UDP does not require any labeled source data so that researchers have proposed UDP to avoid a CPDP limitation of different distributions between source and target datasets. Recently, fully automated unsupervised defect prediction approaches have been proposed by Nam and Kim [66] and Zhang et al. [76]. In the experiments, we chose to use CLAMI proposed by Nam and Kim [66] for UDP because of the following reasons. First, there are no

2. Hereafter a rightward arrow (⇒) denotes a prediction combination.

comparative studies between CLAMI and the approach of Zhang et al. yet [66], [76]. Thus, it is difficult to judge which approach is better at this moment. Second, our HDP experimental framework is based on Java and Weka as CLAMI does. This would be beneficial when we compare CLAMI and HDP under the consistent experimental setting. CLAMI conducts its own metric and instance selection heuristics to generate prediction models [66].

5.4 Experimental Design

For the machine learning algorithm, we use seven widely used classifiers such as Simple logistic, Logistic regression, Random Forest, Bayesian Network, Support vector machine, J48 decision tree, and Logistic model tree [1], [7], [8], [77], [77], [78], [79]. For these classifiers, we use Weka implementation with default options [62].

For WPDP, it is necessary to split datasets into training and test sets. We use the two-fold cross validation (CV), which is widely used in the evaluation of defect prediction models [8], [80], [81]. In the two-fold CV, we use one half of the instances for training a model and the rest for test (round 1). Then, we use the two splits (folds) in a reverse way, where we use the previous test set for training and the previous training set for test (round 2). We repeat these two rounds 500 times, i.e., 1,000 tests, since there is randomness in selecting instances for each split [82]. When conducting the two-fold CV, the stratified CV that keeps the buggy rate of the two folds same as that of the original datasets is applied as we used the default options in Weka [62].

For CPDP-CM, CPDP-IFS, UDP, and HDP, we test the model on the same test splits used in WPDP. For CPDP-CM, CPDP-IFS, and HDP, we build a prediction model by using a source dataset, while UDP does not require any source datasets as it is based on the unsupervised learning. Since there are 1,000 different test splits for a within-project prediction, the CPDP-CM, CPDP-IFS, UDP, and HDP models are tested on 1000 different test splits as well.

These settings for comparing HDP to the baselines are for RQ1. The experimental settings for RQ2 is described in Section 7 in detail.

5.5 Measures

To evaluate the prediction performance, we use the area under the receiver operating characteristic curve (AUC). Evaluation measures such as precision is highly affected by prediction thresholds and defective ratios (class imbalance) of datasets [83]. However, the AUC is known as a useful measure for comparing different models and is widely used because AUC is unaffected by class imbalance as well as being independent from the cutoff probability (prediction threshold) that is used to decide whether an instance should be classified as positive or negative [11], [78], [79], [83], [84]. Mende confirmed that it is difficult to compare the defect prediction performance reported in the defect prediction literature since prediction results come from the different cutoffs of prediction thresholds [85]. However, the receiver operating characteristic curve is drawn by both the true positive rate (recall) and the false positive rate on various prediction threshold values. The higher AUC represents better prediction performance and the AUC of 0.5 means the performance of a random predictor [11].

To measure the effect size of AUC results among baselines and HDP, we compute Cliff's δ that is a non-parametric effect size measure [86]. As Romano et al. suggested, we evaluate the magnitude of the effect size as follows: negligible ($|\delta| < 0.147$), small ($|\delta| < 0.33$), medium ($|\delta| < 0.474$), and large ($0.474 \leq |\delta|$) [86].

To compare HDP by our approach to baselines, we also use the Win/Tie/Loss evaluation, which is used for performance comparison between different experimental settings in many studies [87], [88], [89]. As we repeat the experiments 1,000 times for a target project dataset, we conduct the Wilcoxon signed-rank test ($p < 0.05$) for all AUC values in baselines and HDP [90]. If an HDP model for the target dataset outperforms a corresponding baseline result after the statistical test, we mark this HDP model as a 'Win'. In a similar way, we mark an HDP model as a 'Loss' when the results of a baseline are better than those of our HDP approach with statistical significance. If there is no difference between a baseline and HDP with statistical significance, we mark this case as a 'Tie'. Then, we count the number of wins, ties, and losses for HDP models. By using the Win/Tie/Loss evaluation, we can investigate how many HDP predictions it will take to improve baseline approaches.

6 Prediction Performance of HDP

In this section, we present the experimental results of the HDP approach to address RQ1.

RQ1: Is heterogeneous defect prediction comparable to WPDP, existing CPDP approaches for heterogeneous metric sets (CPDP-CM and CPDP-IFS), and UDP?

RQ1 leads us to investigate whether our HDP is comparable to WPDP (Baseline1), CPDP-CM (Baseline2), CDDP-IFS (Baseline3), and UDP (Baseline4). We report the representative HDP results in Sections 6.1, 6.2, 6.3, and 6.4 based on Gain ratio attribute selection for metric selection, KSAnalyzer with the cutoff threshold of 0.05, and the Logistic classifier. Among different metric selections, Gain ratio attribute selection with Logistic led to the best prediction performance overall. In terms of analyzers, KSAnalyzer led to the best prediction performance. Since the KSAnalyzer is based on the p-value of a statistical test, we chose a cutoff of 0.05 which is one of commonly accepted significance levels in the statistical test [91].

In Sections 6.5, 6.6, and 6.7, we report the HDP results by using various metric selection approaches, metric matching analyzers, and machine learners respectively to investigate HDP performances more in terms of RQ1.

6.1 Comparison Result with Baselines

Table 2 shows the prediction performance (a median AUC) of baselines and HDP by KSAnalyzer with the cutoff of 0.05 and Cliff's δ with its magnitude for each target. The last row, *All* targets, show an overall prediction performance of baslines and HDP in a median AUC. Baseline1 represents the WPDP results of a target project and Baseline2 shows the CPDP results using common metrics (CPDP-CM) between source and target projects. Baseline3 shows the results of CPDP-IFS proposed by He et al. [56] and Baseline4 represents the UDP results by CLAMI [66]. The last column shows the HDP results by KSAnalyzer with the cutoff of 0.05. If there are better results between Baseline1 and our

TABLE 2
Comparison Results Among WPDP, CPDP-CM, CPDP-IFS, UDP, and HDP by KSAnalyzer with the Cutoff of 0.05 in a Median AUC

Target	WPDP (Baseline1)	CPDP-CM (Baseline2)	CPDP-IFS (Baseline3)	UDP (Baseline4)	HDP KS
EQ	**0.801** (-0.519,L)	0.776 (-0.126,**N**)	0.461 (0.996,**L**)	0.737 (0.312,**S**)	0.776*
JDT	**0.817** (-0.889,L)	0.781 (0.153,**S**)	0.543 (0.999,**L**)	0.733 (0.469,**M**)	0.767*
LC	**0.765** (-0.915,L)	0.636 (0.059,**N**)	0.584 (0.198,**S**)	0.732& (-0.886,L)	0.655
ML	**0.719** (-0.470,M)	0.651 (0.642,**L**)	0.557 (0.999,**L**)	0.630 (0.971,**L**)	0.692*&
PDE	**0.731** (-0.673,L)	0.681 (0.064,**N**)	0.566 (0.836,**L**)	0.646 (0.494,**L**)	0.692*
Apache	**0.757** (-0.398,M)	0.697 (0.228,**S**)	0.618 (0.566,**L**)	0.754& (-0.404,M)	0.720*
Safe	0.829 (-0.002,**N**)	0.749 (0.409,**M**)	0.630 (0.704,**L**)	0.773 (0.333,**M**)	0.837*&
ZXing	0.626 (0.409,**M**)	0.618 (0.481,**L**)	0.556 (0.616,**L**)	0.644 (0.099,**N**)	**0.650**
ant-1.3	0.800 (-0.211,S)	0.781 (0.163,**S**)	0.528 (0.579,**L**)	0.775 (-0.069,**N**)	0.800*
arc	0.726 (-0.288,S)	0.626 (0.523,**L**)	0.547 (0.954,**L**)	0.615 (0.677,**L**)	0.701
camel-1.0	0.722 (-0.300,S)	0.590 (0.324,**S**)	0.500 (0.515,**L**)	0.658 (-0.040,**N**)	0.639
poi-1.5	0.717 (-0.261,S)	0.675 (0.230,**S**)	0.640 (0.509,**L**)	0.720 (-0.307,S)	0.706
redaktor	**0.719** (-0.886,L)	0.496 (0.067,**N**)	0.489 (0.246,**S**)	0.489 (0.184,**S**)	0.528
skarbonka	0.589 (0.594,**L**)	0.744 (-0.083,**N**)	0.540 (0.581,**L**)	0.778& (-0.353,M)	**0.694***
tomcat	**0.814** (-0.935,L)	0.675 (0.961,**L**)	0.608 (0.999,**L**)	0.725 (0.273,**S**)	0.737*&
velocity-1.4	**0.714** (-0.987,L)	0.412 (-0.142,**N**)	0.429 (-0.138,**N**)	0.428 (-0.175,S)	0.391
xalan-2.4	**0.772** (-0.997,L)	0.658 (-0.997,L)	0.499 (0.894,**L**)	0.712& (-0.998,L)	0.560*
xerces-1.2	0.504 (-0.040,**N**)	0.462 (0.446,**M**)	0.473 (0.200,**S**)	0.456 (0.469,**M**)	0.497
cm1	**0.741** (-0.383,M)	0.597 (0.497,**L**)	0.554 (0.715,**L**)	0.675 (0.265,**S**)	0.720*
mw1	0.726 (-0.111,**N**)	0.518 (0.482,**L**)	0.621 (0.396,**M**)	0.680 (0.236,**S**)	0.745
pc1	**0.814** (-0.668,L)	0.666 (0.814,**L**)	0.557 (0.997,**L**)	0.693 (0.866,**L**)	0.754*&
pc3	**0.790** (-0.819,L)	0.665 (0.815,**L**)	0.511 (1.000,**L**)	0.667 (0.921,**L**)	0.738*&
pc4	**0.850** (-1.000,L)	0.624 (0.204,**S**)	0.590 (0.856,**L**)	0.664 (0.287,**S**)	0.681*
jm1	**0.705** (-0.662,L)	0.571 (0.662,**L**)	0.563 (0.914,**L**)	0.656 (0.665,**L**)	0.688*
pc2	0.878 (0.202,**S**)	0.634 (0.795,**L**)	0.474 (0.988,**L**)	0.786 (0.996,**L**)	0.893*&
pc5	0.932 (0.828,**L**)	0.841 (0.999,**L**)	0.260 (0.999,**L**)	0.885 (0.999,**L**)	**0.950***&
mc1	0.885 (0.164,**S**)	0.832 (0.970,**L**)	0.224 (0.999,**L**)	0.806 (0.999,**L**)	**0.893***&
mc2	0.675 (-0.003,**N**)	0.536 (0.675,**L**)	0.515 (0.592,**L**)	0.681 (-0.096,**N**)	0.682*
kc3	0.647 (0.099,**N**)	0.636 (0.254,**S**)	0.568 (0.617,**L**)	0.621 (0.328,**S**)	0.678*
ar1	0.614 (0.420,**M**)	0.464 (0.647,**L**)	0.586 (0.398,**M**)	0.680 (0.213,**S**)	0.735
ar3	0.732 (0.356,**M**)	0.839 (0.243,**S**)	0.664 (0.503,**L**)	0.750 (0.343,**M**)	**0.830***&
ar4	0.816 (-0.076,**N**)	0.588 (0.725,**L**)	0.570 (0.750,**L**)	0.791 (0.139,**N**)	0.805*&
ar5	0.875 (0.043,**N**)	0.875 (0.287,**S**)	0.766 (0.339,**M**)	0.893 (-0.037,**N**)	0.911
ar6	**0.696** (-0.149,S)	0.613 (0.377,**M**)	0.524 (0.485,**L**)	0.683 (-0.133,**N**)	0.676*
All	**0.732**	0.632	0.558	0.702	0.711*&

(Cliff's δ magnitude — N: Negligible, S: Small, M: Medium, and L: Large).

approach with statistical significance (Wilcoxon signed-rank test [90], $p < 0.05$), the better AUC values are in bold font as shown in Table 2. Between Baseline2 and our approach, better AUC values with statistical significance are underlined in the table. Between Baseline3 and our approach, better AUC values with statistical significance are shown with an asterisk (*). Between Baseline4 and our approach, better AUC values with statistical significance are shown with an ampersand (&).

The values in parentheses in Table 2 show Cliff's δ and its magnitude for the effect size among baselines and HDP. If a Cliff's δ is a positive value, HDP improves a baseline in terms of the effect size. As explained in Section 5.5, based on a Cliff's δ, we can estimate the magnitude of the effect size (N: Negligible, S: Small, M: Medium, and L: Large). For example, the Cliff's δ of AUCs between WPDP and HDP for pc5 is 0.828 and its magnitude is *Large* as in Table 2. In other words, HDP outperforms WPDP in pc5 with the large magnitude of the effect size.

We observed the following results about RQ1:

- The 18 out of 34 targets (Safe, ZXing, ant-1.3, arc, camel-1.0, poi-1.5, skarbonka, xerces-1.2, mw1, pc2, pc5, mc1, mc2, kc3, ar1, ar3, ar4, and ar5) show better with statistical significance or comparable results against WPDP. However, HDP by KSAnalyzer with the cutoff of 0.05 did not lead to better with statistical significance or comparable against WPDP in *All* in our empirical settings. Note that WPDP is an upper bound of prediction performance. In this sense, HDP shows potential when there are no training datasets with the same metric sets as target datasets.
- The Cliff's δ values between WPDP and HDP are positive in 14 out of 34 targets. In about 41 percent targets, HDP shows negligible or better results to HDP in terms of effect size.
- HDP by KSAnalyzer with the cutoff of 0.05 leads to better or comparable results to CPDP-CM with statistical significance. (no underlines in CPDP-CM of Table 2)
- HDP by KSAnalyzer with the cutoff of 0.05 outperforms CPDP-CM with statistical significance when considering results from *All* targets in our experimental settings.
- The Cliff's δ values between CPDP-CM and HDP are positive in 30 out 34 targets. In other words, HDP improves CPDP-CM in most targets in terms of effect size.

TABLE 3
Median AUCs of Baselines and HDP in KSAnalyzer (Cutoff = 0.05) by Each Source Group

Source	WPDP	CPDP-CM	CPDP-IFS	UDP	HDP KS,0.05	HDP Target Coverage
AEEEM	0.732	0.750	0.722	0.776	0.753	35%
Relink	**0.731**	0.655	0.500	0.683	0.694*	84%
MORPH	0.741	0.652	0.589	0.732	0.724*	92%
NASA	0.732	0.550	0.541	0.754	0.734*	100%
SOFTLAB	**0.741**	0.631	0.551	0.681	0.692*	100%

- HDP by KSAnalyzer with the cutoff of 0.05 leads to better or comparable results to CPDP-IFS with statistical significance. (no asterisks in CPDP-IFS of Table 2)
- HDP by KSAnalyzer with the cutoff of 0.05 outperforms CPDP-IFS with statistical significance when considering results from *All* targets in our experimental settings.
- The Cliff's δ values between CPDP-IFS and HDP are positive in all targets except for velocity-1.4.
- HDP by KSAnalyzer with the cutoff of 0.05 outperforms UDP with statistical significance when considering results from *All* targets in our experimental settings.
- The magnitude of Cliff's δ values between UDP and HDP are negligible or positively better in 29 out 34 targets.

6.2 Target Prediction Coverage

Target prediction coverage shows how many target projects can be predicted by the HDP models. If there are no feasible prediction combinations for a target because of there being no matched metrics between source and target datasets, it might be difficult to use an HDP model in practice.

For target prediction coverage, we analyzed our HDP results by KSAnalyzer with the cutoff of 0.05 by each source group. For example, after applying metric selection and matching, we can build a prediction model by using EQ in AEEEM and predict each of 29 target projects in four other dataset groups. However, because of the cutoff value, some predictions may not be feasible. For example, EQ⇒Apache was not feasible because there are no matched metrics whose matching scores are greater than 0.05. Instead, another source dataset, JDT, in AEEEM has matched metrics to Apache. In this case, we consider the source group, AEEEM, covered Apache. In other words, if any dataset in a source group can be used to build an HDP model for a target, we count the target prediction is as covered.

Table 3 shows the median AUCs and prediction target coverage. The median AUCs were computed by the AUC values of the feasible HDP predictions and their corresponding predictions of WPDP, CPDP-CM, CPDP-IFS, and UDP. We conducted the Wilcoxon signed-rank test on results between WPDP and baselines [90]. Like Table 2, better results between baselines and our approach with statistical significance are in bold font, underlined, with asterisks and/or with ampersands.

First of all, in each source group, we could observe WPDP did not outperform HDP in three source groups, AEEEM, MORPH, and NASA, with statistical significance. For example, 29 target projects ($34-5$ AEEEM datasets) were predicted by some projects in AEEEM and the median AUC for HDP by KSAnalyzer is 0.753 while that of WPDP is 0.732. In addition, HDP by KSAnalyzer also outperforms CPDP-CM and CPDP-IFS. There are no better results in CPDP-CM than those in HDP by KSAnalyzer with statistical significance (no underlined results in third column in Table 3). In addition, HDP by KSAnalyzer outperforms CPDP-IFS in most source groups with statistical significance except for AEEEM. Between UDP and HDP, we did not observe significant performance difference as there are no ampersands in any AUC values in both UDP and HDP.

The target prediction coverage in the NASA and SOFTLAB groups yielded 100 percent as shown in Table 3. This implies our HDP models may conduct defect prediction with high target coverage even using datasets which only appear in one source group. AEEEM, ReLink, and MORPH groups have 35, 84, and 92 percent respectively since some prediction combinations do not have matched metrics because of low matching scores ($\leq$0.05). Thus, some prediction combinations using matched metrics with low matching scores can be automatically excluded. In this sense, our HDP approach follows a similar concept to the two-phase prediction model [92]: (1) checking prediction feasibility between source and target datasets, and (2) predicting defects. This mechanism is helpful to filter out the matched metrics whose distributions are not similar depending on a matching score.

Target coverage limitation from AEEEM, ReLink, or MORPH groups can be solved by using either NASA or SOFTLAB groups. This shows the scalability of HDP as it can easily overcome the target coverage limitation by adding any existing defect datasets as a source until we can achieve the 100 percent target coverage.

6.3 Win/Tie/Loss Results

To investigate the evaluation results for HDP in detail, we report the Win/Tie/Loss results of HDP by KSAnalyzer with the cutoff of 0.05 against WPDP (Baseline1), CPDP-CM (Baseline2), CPDP-IFS (Baseline3), and UDP (Baseline4) in Table 4.

KSAnalyzer with the cutoff of 0.05 conducted 284 out of 962 prediction combinations since 678 combinations do not have any matched metrics because of the cutoff threshold. In Table 4, the target dataset, ZXing, was predicted in five prediction combinations and our approach, HDP, outperforms Baselines in the four or five combinations (i.e., 4 or 5 Wins). However, CPDP-CM and CPDP-IFS outperform HDP in one combination of the target, ZXing (1 Loss).

Against Baseline1, the four targets such as ZXing, skarbonka, pc5, and mc1 have only Win results. In other words, defects in those four targets could be predicted better by other source projects using HDP models by KSAnalyzer compared to WPDP models.

In Fig. 4, we analyzed distributions of matched metrics using box plots for one of Win cases, ant-1.3⇒ar5. The gray, black, and white box plots show distributions of matched metric values in all, buggy, and clean instances respectively. The three box plots on the left-hand side represent distributions of a source metric while the three box plots on the

TABLE 4
Win/Tie/Loss Results of HDP by KSAnalyzer (Cutoff = 0.05) Against WPDP (Baseline1), CPDP-CM (Baseline2), CPDP-IFS (Baseline3), and UDP (Baseline4)

Target	Against											
	WPDP (Baseline1)			CPDP-CM (Baseline2)			CPDP-IFS (Baseline3)			UDP (Baseline4)		
	Win	Tie	Loss	Win	Tie	Loss	Win	Tie	Loss	Win	Tie	Loss
EQ	0	0	4	1	2	1	4	0	0	3	0	1
JDT	0	0	5	3	0	2	5	0	0	4	0	1
LC	0	0	7	3	3	1	3	1	3	0	0	7
ML	0	0	6	4	2	0	6	0	0	6	0	0
PDE	0	0	5	2	0	3	5	0	0	4	0	1
Apache	4	0	8	8	0	4	10	0	2	4	0	8
Safe	11	1	7	14	0	5	17	0	2	15	1	3
ZXing	5	0	0	4	0	1	4	0	1	4	1	0
ant-1.3	5	1	5	7	0	4	9	0	2	6	0	5
arc	0	0	3	2	0	1	3	0	0	3	0	0
camel-1.0	2	0	5	5	0	2	6	0	1	3	0	4
poi-1.5	2	0	2	3	1	0	2	0	2	2	0	2
redaktor	0	0	4	2	0	2	3	0	1	3	0	1
skarbonka	15	0	0	5	1	9	13	0	2	2	0	13
tomcat	0	0	1	1	0	0	1	0	0	1	0	0
velocity-1.4	0	0	6	2	0	4	2	0	4	2	0	4
xalan-2.4	0	0	1	0	0	1	1	0	0	0	0	1
xerces-1.2	1	0	1	2	0	0	1	0	1	2	0	0
cm1	0	1	9	8	0	2	9	0	1	7	0	3
mw1	4	0	3	5	0	2	5	0	2	5	0	2
pc1	0	0	7	6	0	1	7	0	0	7	0	0
pc3	0	0	7	7	0	0	7	0	0	7	0	0
pc4	0	0	8	5	0	3	8	0	0	6	0	2
jm1	1	0	5	5	0	1	6	0	0	5	0	1
pc2	4	0	1	5	0	0	5	0	0	5	0	0
pc5	1	0	0	1	0	0	1	0	0	1	0	0
mc1	1	0	0	1	0	0	1	0	0	1	0	0
mc2	10	2	6	15	0	3	14	0	4	8	2	8
kc3	9	0	2	8	0	3	10	0	1	9	0	2
ar1	12	0	2	12	1	1	10	0	4	12	0	2
ar3	15	0	2	8	0	9	11	2	4	15	0	2
ar4	6	1	10	15	1	1	16	0	1	13	2	2
ar5	15	0	7	15	0	7	15	0	7	14	1	7
ar6	5	0	11	10	3	3	13	0	3	5	3	8
Total	128	6	150	194	14	76	233	3	48	184	10	90
%	45.1%	2.1%	52.8%	68.3%	4.9%	26.8%	82.0%	1.1%	16.9%	64.8%	3.5%	31.7%

right-hand side represent those of a target metric. The bottom and top of the boxes represent the first and third quartiles respectively. The solid horizontal line in a box represents the median metric value in each distribution. Black points in the figure are outliers.

Fig. 4 explains how the prediction combination of ant-1.3⇒ar5 can have a high AUC, 0.946. Suppose that a simple model predicts that an instance is buggy when the metric value of the instance is more than 40 in the case of Fig. 4. In both datasets, approximately 75 percent or more buggy and clean instances will be predicted correctly. In Fig. 4, the matched metrics in ant-1.3⇒ar5 are the response for class (*RFC*: number of methods invoked by a class) [93] and the number of unique operands (*unique_operands*) [4], respectively. The *RFC* and *unique_operands* are not the same metric so it might look like an arbitrary matching. However, they are matched based on their similar distributions as shown in Fig. 4. Typical defect prediction metrics have tendencies in which higher complexity causes more defect-proneness [1], [2], [6]. In Fig. 4, instances with higher values of *RFC* and *unique_operands* have the tendency to be more defect-prone. For this reason, the model using the matched metrics could achieve such a high AUC (0.946). We could observe this defect-proneness tendency in other Win results (See the online appendix, https://lifove.github.io/hdp/#pc). Since matching metrics is based on similarity of source and target metric distributions, HDP also addresses several issues

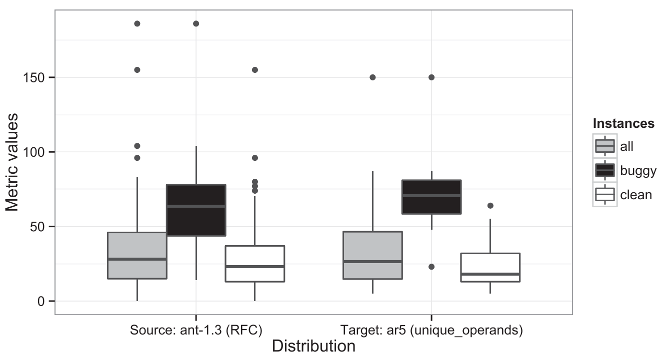


Fig. 4. Distribution of metrics (matching score=0.91) from ant-1.3⇒ar5 (AUC = 0.946).

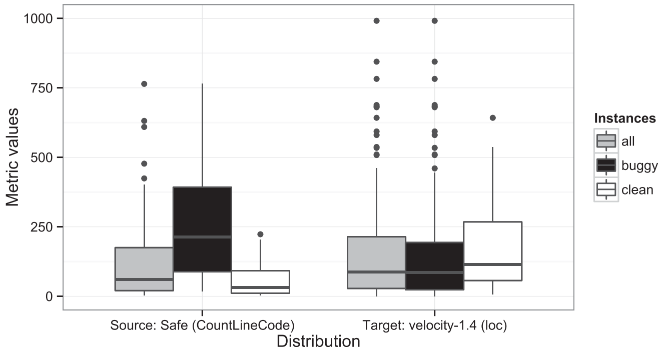


Fig. 5. Distribution of metrics (matching score=0.45) from Safe⇒velocity-1.4 (AUC = 0.391).

related to a dataset shift such as the covariate shift and domain shift discussed by Turhan [94].

However, there are still about 52.8 percent Loss results against WPDP as shown in Table 4. The 14 targets have no Wins at all against Baseline1. In addition, other targets still have Losses even though they have Win or Tie results.

As a representative Loss case, we investigated distributions of the matched metrics in Safe⇒velocity-1.4, whose AUC is 0.391. As observed, Loss results were usually caused by different tendencies of defect-proneness between source and target metrics. Fig. 5 shows how the defect-prone tendencies of source and target metrics are different. Interestingly, the matched source and target metric by the KSAnalyzer is the same as *LOC* (*CountLineCode* and *loc*) in both. As we observe in the figure, the median metric value of buggy instances is higher than that of clean instances in that the more *LOC* implies the higher defect-proneness in the case of Safe. However, the median metric value of buggy instances in the target is lower than that of clean instances in that the less *LOC* implies the higher defect-proneness in velocity-1.4. This inconsistent tendency of defect-proneness between the source and target metrics could degrade the prediction performance although they are the same metric.

We regard the matching that has an inconsistent defect-proneness tendency between source and target metrics as *a noisy metric matching*. We could observe this kind of noisy metric matching in prediction combinations in other Loss results.

However, it is very challenging to filter out the noisy metric matching since we cannot know labels of target instances in advance. If we could design a filter for the noisy metric matching, the Loss results would be minimized. Thus, designing a new filter to mitigate these Loss results is an interesting problem to address. Investigating this new filter for the noisy metric matching will remain as future work.

Fig. 5 also explains why CPDP-CM did not show reasonable prediction performance. Although the matched metrics are same as *LOC*, its defect-prone tendency is inconsistent. Thus, this matching using the common metric was noisy and was not helpful for building a prediction model.

Overall, the numbers of Win and Tie results are 128 and 6 respectively out of all 284 prediction combinations. This means that in 47.1 percent of prediction combinations our HDP models achieve better or comparable prediction performance than those in WPDP.

TABLE 5
HDP Prediction Performance in Median AUC by Source Datasets

Source	AUC	# of Targets	Source	AUC	# of Targets
EQ	0.794	5	JDT	0.756	10
LC	0.674	2	ML	0.714	3
PDE	n/a	0	Apache	0.720	17
Safe	0.684	22	ZXing	0.707	12
ant-1.3	0.738	16	arc	0.666	8
camel-1.0	0.803	2	poi-1.5	0.761	6
redaktor	n/a	0	skarbonka	0.692	17
tomcat	0.739	9	velocity-1.4	n/a	0
xalan-2.4	0.762	7	xerces-1.2	n/a	0
cm1	0.630	9	mw1	0.710	13
pc1	0.734	9	pc3	0.786	9
pc4	n/a	0	jm1	0.678	8
pc2	0.822	3	pc5	n/a	0
mc1	0.856	3	mc2	0.739	20
kc3	0.689	5	ar1	0.320	3
ar3	0.740	11	ar4	0.674	18
ar5	0.691	28	ar6	0.740	9

However, HDP shows relatively better results against Baseline2, Baseline3, and Baseline4 in terms of the Win/Tie/Loss evaluation. In the 208 (73.2 percent) out of 284 prediction combinations, HDP outperforms and is comparable to CPDP-CM. Against Baseline3, 236 (83.1 percent) prediction combinations are Win or Tie results. Against Baseline4, HDP has 194 Win or Tie results (68.3 percent). In addition, there are at least one Win case for all targets against CPDP-CM, CPDP-IFS, and UDP except for LC and xalan-2.4 in UDP (Table 4). From 284 out of 962 combinations, we could achieve the 100 percent target coverage and find at least one HDP model that are better than that by CPDP-CM, CPDP-IFS, or UDP in most combinations.

6.4 Performance by Source Datasets

Table 5 shows prediction performance of HDP (KSAnalyzer, cutoff=0.05, and Gain Ratio feature selection) by each source dataset. The 3rd and 6th columns represent the number of targets predicted by a source. For example, EQ predicts five targets by HDP and the median AUC from these five target predictions is 0.794. Since the total number of feasible target predictions is 284 out of all 962 prediction combinations, six source datasets (PDE, redaktor, velocity-1.4, xerces-1.2, pc4, and pc5) did not predict any targets because there were no matched metrics.

The higher defect ratio of a training dataset may make bias as the prediction performance of the dataset with the higher defect ratio may be better than that with the lower defect ratio [83]. To investigate if HDP is also affected by defect ratio of the training dataset and what makes better prediction performance, we analyzed the best and worst source datasets that lead to the best and worst AUC values, respectively.

We found that HDP does not bias prediction performance from the defect ratios of datasets and prediction performance is highly depending on the defect-proneness tendency of matched metrics under our experiments. As shown in Table 5, the best source dataset is mc1 (0.856) although its defect ratio is very low, 0.73 percent. We

TABLE 6
Prediction Performance (a Median AUC and % of Win) in Different Metric Selections

Approach	Against								HDP
	WPDP		CPDP-CM		CPDP-IFS		UDP		
	AUC	Win%	AUC	Win%	AUC	Win%	AUC	Win%	AUC
Gain Ratio	**0.732**	45.1%	0.632	68.3%	0.558	82.0%	0.702	64.8%	0.711*&
Chi-Square	**0.741**	43.0%	0.635	77.5%	0.557	83.3%	0.720&	65.2%	0.717*
Significance	**0.734**	43.8%	0.630	69.7%	0.557	83.4%	0.693	67.6%	0.713*&
Relief-F	**0.740**	42.4%	0.642	66.2%	0.540	80.8%	0.720	62.6%	0.706*
None	0.657	46.7%	0.622	51.6%	0.545	64.2%	0.693&	44.0%	**0.665***

TABLE 7
Prediction Performance in Other Analyzers with the Matching Score Cutoffs, 0.05 and 0.90

Analyzer	Cutoff	Against								HDP	Target Coverage	# of Prediction Combination
		WPDP		CPDP-CM		CPDP-IFS		UDP				
		AUC	Win%	AUC	Win%	AUC	Win%	AUC	Win%	AUC		
P	0.05	**0.741**	43.0%	0.655	54.9%	0.520	69.5%	0.693&	69.5%	0.642*	100%	962
P	0.90	**0.732**	32.9%	0.629	62.9%	0.558	80.0%	0.680	59.3%	0.693*	100%	140
KS	0.05	**0.732**	45.1%	0.632	68.3%	0.558	82.0%	0.702	64.8%	0.711*&	100%	284
KS	0.90	0.816	55.6%	0.588	77.8%	0.585	100.0%	0.786	88.9%	0.831*	21%	90
SCo	0.05	**0.741**	16.9%	0.655	41.9%	0.520	55.7%	0.693&	38.7%	0.609*	100%	962
SCo	0.90	**0.741**	17.7%	0.654	42.4%	0.520	56.4%	0.693&	39.2%	0.614*	100%	958

investigate distributions of matched metrics of EQ like Figs. 4 and 5.[3] We observed that all the matched metrics for the source, EQ, show the typical defect-proneness tendency similarly to Fig. 4. The worst source dataset is ar1 (0.320) whose defect ratio is 7.44 percent. We observed that the matched metrics of ar1 show inconsistent tendency of defect-proneness between source and target, i.e., noisy metric matching. From these best and worst cases, we confirm again the consistent defect-proneness tendency of matched metrics between source and target datasets is most important to lead to better prediction performance.

6.5 Performance in Different Metric Selections

Table 6 shows prediction results on various metric selection approaches including with no metric selection ('None'). We compare the median AUCs of the HDP results by KSAnalyzer with the cutoff of 0.05 to those of WPDP, CPDP-CM, CPDP-IFS, or UDP and report the percentages of Win results.

Overall, we could observe metric selection to be helpful in improving prediction models in terms of AUC. When applying metric selection, the Win results account for more than about 63 percent in most cases against CPDP-CM and UDP. Against CPDP-IFS, the Win results of HDP account for more than 80 percent after applying the metric selection approaches. This implies that the metric selection approaches can remove irrelevant metrics to build a better prediction model. However, the percentages of Win results in 'None' were lower than those in applying metric selection. Among metric selection approaches, 'Gain Ratio', 'Chi-Square' and 'Significance' based approaches lead to the best performance in terms of the percentages of the Win results (64.8-83.4 percent) against CPDP-CM, CPDP-IFS, and UDP.

3. For detailed target prediction results and distributions of matched metrics by each source dataset, please refer to the online appendix: https://lifove.github.io/hdp/#pc.

6.6 Performance in Various Metric Matching Analyzers

In Table 7, we compare the prediction performance in other analyzers with the matching score cutoff thresholds, 0.05 and 0.90. HDP's prediction results by PAnalyzer, with a cutoff of 0.90, are comparable to CPDP-CM and CPDP-IFS. This implies that comparing 9 percentiles between source and target metrics can evaluate the similarity of them well with a threshold of 0.90 against CPDP-CM and CPDP-IFS. However, PAnalyzer with the cutoff is too simple to lead to better prediction performance than KSAnalyzer. In KSAnalyzer with a cutoff of 0.05, the AUC (0.711) better than that (0.693) of PAnalyzer with the cutoff of 0.90.

HDP by KSAnalyzer with a cutoff of 0.90 could show better AUC value (0.831) compared to that (0.711) with the cutoff of 0.05. However, the target coverage is just 21 percent. This is because some prediction combinations are automatically filtered out since poorly matched metrics, whose matching score is not greater than the cutoff, are ignored. In other words, defect prediction for 79 percent of targets was not conducted since the matching scores of matched metrics in prediction combinations for the targets are not greater than 0.90 so that all matched metrics in the combinations were ignored.

An interesting observation in PAnalyzer and KSAnalyzer is that AUC values of HDP by those analyzers tend to be improved when a cutoff threshold increased. As the cutoff threshold increased as 0.05, 0.10, 0.20, ..., and 0.90, we observed prediction results by PAnalyzer and KSAnalyzer gradually are improved from 0.642 to 0.693 and 0.711 to 0.831 in AUC, respectively. This means these two analyzers

TABLE 8
Prediction Performance (a Median AUC and % of Win) of HDP by KSAnalyzer (Cutoff = 0.05) Against WPDP, CPDP-CM, and CPDP-IFS by Different Machine Learners

HDP Learners	Against								HDP
	WPDP		CPDP-CM		CPDP-IFS		UDP		
	AUC	Win	AUC	Win	AUC	Win	AUC	Win	AUC
SimpleLogistic	0.763	44.0%	0.680	60.9%	0.691	62.7%	0.734	48.9%	0.718*
RandomForest	**0.732**	39.4%	0.629	46.5%	0.619	63.0%	0.674&	84.9%	0.640*
BayesNet	**0.703**	41.5%	0.583	48.2%	0.675*	29.2%	0.666&	35.2%	0.633
SVM	0.500	29.9%	0.500	28.2%	0.500	26.4%	0.635&	11.6%	0.500
J48	**0.598**	34.2%	0.500	44.7%	0.558	46.8%	0.671&	18.7%	0.568
Logistic	**0.732**	45.1%	0.632	68.3%	0.558	82.0%	0.702	64.8%	0.711*&
LMT	**0.751**	42.3%	0.671	58.5%	0.690	56.0%	0.734&	41.9%	0.702

can filter out negative prediction combinations well. As a result, the percentages of Win results are also increased.

HDP results by SCoAnalyzer were worse than WPDP, CPDP-CM, and UDP. In addition, prediction performance rarely changed regardless of cutoff thresholds; results by SCoAnalyzer in different cutoffs from 0.05 to 0.90 did not vary as well. A possible reason is that SCoAnalyzer does not directly compare the distributions between source and target metrics. This result implies that the similarity of distribution between source and target metrics is a very important factor for building a better prediction model.

6.7 Performance in Various Machine Learners

To investigate if HDP works with other machine learners, we built HDP models (KSAnalyzer and the cutoff of 0.05) with various learners used in defect prediction literature such as SompleLogistic, Random Forest, BayesNet, SVM, J48 Decision Tree, and Logistic Model Trees (LMT) [1], [7], [8], [77], [77], [78], [79]. Table 8 shows median AUCs and Win results.

Machine learners based on logit function such as SimpleLogistic, Logistic, and LMT led to the promising results among various learners (median AUC $>$ 0.70). Logistic Regression uses the logit function and Simple Logistic builds a linear logistic regression model based on LogitBoost [95]. LMT adopts Logistic Regression at the leaves of decision tree [77]. Thus, these learners work well when there is a linear relationship between a predictor variable (a metric) and the logit transformation of the outcome variable (defect-proneness) [95], [96]. In our study, this linear relationship is related to the defect-proneness tendency of a metric, that is, a higher complexity causes more defect-proneness [1], [2], [6]. As the consistent defect-prone tendency of matched metrics is important in HDP, the HDP models built by the logistic-based learners can lead to the promising prediction performance.

According to the recent study by Ghotra et al., LMT and Simple Logistic tend to lead to better prediction performance than other kinds of machine learners [77]. HDP results based on Simple Logistic and LMT also confirm the results by Ghotra et al. [77]. However, these results do not generalize HDP works best by logistic-based learners as Ghotra et al. also pointed out prediction results and the best machine learner may vary based on each dataset [77].

There are several interesting observations in Table 8. SVM did not work for HDP and all baselines as their AUC values are 0.500. This result also confirms the study by Ghotra et al. as SVM was ranked in the lowest group [77]. Except for SimpleLogistic and Logistic, UDP outperforms HDP in most learners with statistical significance. CLAMI for UDP is also based on defect-proneness tendency of a metric [66]. If target datasets follow this tendency very well, CLAMI could lead to promising prediction performance as CLAMI is not affected by the distribution differences between source and target datasets [66]. Detailed comparison of UDP and HDP is an interesting future direction as UDP techniques have received much attention recently [66], [76], [97]. We remain this detailed comparative study as future work.

6.8 Summary

In Section 6, we showed HDP results for RQ1. The followings are the key observations of the results in our experimental setting:

- Overall, HDP led to better or comparable results to the baselines such as CPDP-CM, CPDP-IFS, and UDP when using the Logistic learner with KSAnalyzer (the cutoff of 0.05) and Gain ratio attribute selection.
- Compared to WPDP (0.732), HDP achieved 0.711 in terms of median AUC. Note that WPDP is an upper bound and 18 of 34 projects show better prediction results with statistical significance in terms of median AUC. However, there are still 52.8 percent of Loss results against WPDP. Based on the analysis of distributions of matched metrics, we observed that the Loss cases are caused by the inconsistent defect-proneness tendency of the matched metrics. Identifying the inconsistent tendency in advance is a challenging problem to be solved.
- Applying metric selection approaches could improve HDP performances against the baselines.
- KSAnalyzer showed the best HDP performance compared to PAnalyzer and SCoAnalyzer. This confirms that KS-test is a good tool to decide whether distributions of two variables are drawn from the same distribution [67], [70].
- HDP worked well with Simple Logistic, Logistic, and LMT but not with other machine learners. One possible reason is that Logistic related classifiers capture the linear relationship between metrics and the logit transformation of labels that is related to the defect-proneness tendency of the metrics.

In the case of homogeneous transfer (where the source and target datasets have the same variable names), we have results with Krishna et al. [28]. It has shown that within "communities" (projects that collect data using the same variable names) there exists one "bellwether"[4] dataset from which it is possible to learn defect predictors that work well for all other members of that community. (Aside: this also means that within each community there are projects that always produce demonstrably worse defect models.) While such bellwethers are an interesting way to simplify homogeneous transfer learning, our own experiments show that this "bellwether" idea does not work for heterogeneous transfer (where source and target can have different terminology). We conjecture that bellwethers work for homogeneous data due to regularities in the data that may not be present in the heterogeneous case.

7 Size Limits (Lower Bounds) for Effective Transfer Learning

In this section, we investigate the lower bounds of the effective sizes of source and target datasets for HDP models to address RQ2.

RQ2: What are the lower bounds of the size of source and target datasets for effective HDP?

Since HDP compares the distributions of source metrics to those of target metrics, it is important to seek the empirical evidence for the effective sizes of source and target datasets to match source and target metrics. We first present the results of the empirical study for RQ2 in this section and validate the generality of its results in Section 8.

Like prior work [8], [10], [11], [48], [55], the basic HDP method we proposed above uses *all* the instances in potential source and target projects to perform metric to select the best matched metrics and then build defect prediction learners. Collecting *all* that data from source and target projects need much more work and also for the target project, it requires waiting for it to finish before transferring its learned lessons. This begs the question "how early can we transfer?". That is, how *few* historical data and target projects do we need before transfer can be effective? In this section, we conduct an empirical study to answer these questions related to RQ2.

To investigate the size limits for effective transfer learning in the setting of CPDP across datasets with heterogeneous metric sets, we focus on the HDP approach. There are other approaches such as CPDP-IFS [56] and CCA+ [57]. In Section 6, we observed that HDP outperforms CPDP-IFS. In addition, CCA+ was evaluated in somewhat different context, i.e., cross-company defect prediction and with 14 projects which are far less than 34 projects used in our experiments for HDP. In addition, the implementation of CCA+ is not publicly available yet and more complex than HDP. For this reason, we conducted our empirical study for RQ2 based on HDP.

7.1 Using Small Datasets is Feasible

Recall from the above, HDP uses datasets in a two step process. To test the impact of having access to *less* data, we add an instance sampling process before performing metric matching: instead of using all the instances from candidate source and target datasets, those datasets will be randomly sampled (without replacement) to generate smaller datasets of size $N \in \{50, 100, 150, 200\}$.

The reason we choose those N values as follows. On one hand, by looking at the size of datasets used in the above experiment, we observed that minimum size is ar5 with 36 instances and maximum size is pc5 with 17,001 instances. The median and mean value of dataset size are 332 and 1,514, respectively. Then 200 is less than both of them, which is reasonably small. On the other hand, we use these numbers to show using small datasets is feasible compared to the original. We are not claiming they are the best (optimal) small numbers. For most datasets considered in this experiment, N with these values is a small data size compared to the original data size. For example, we use $N \in \{50, 100, 150, 200\}$ whereas our datasets vary in size from 332 to 17,001 (from median to the maximum number of rows).

When sampling the original data, *if the number of instances in the original dataset is smaller than N, all those instances will be included*. For example, $N = 200$ means we sampled both source and target data with size of 200. If the dataset has less than 200 instances, such as ar3, we use all the instances and no oversampling is applied. With those sampled $N = 200$ data, we perform metric matching to build a learner and finally predict labels of all original data in the target project. We sample the data without replacement to avoid duplicate data.

The results for this HDP-with-limited-data experiment is shown in Fig. 6 (we display median AUC results from 20 repeats, using Logistic Regression as the default learner). In that figure:

- The *black* line show the results using *all* data;
- The *colourful* lines show results of transferring from *some* small N number of samples (instead of *all*) in the source and target datasets during metric matching and learner building;
- The letters show the unique ID of each dataset.

The datasets are ordered left to right by the difference to the black line (where we transfer using *all* the source data):

- On the left, the black line is *above* the red line; i.e., for those datasets, we do *better* using *all* data than using *some*.
- On the right, the black line is *below* the red line; i.e., for those datasets, we do *worse*using *all* data than using *some*.

Note that the gap between the red and black line shrinks as we use more data and after $N = 100$, the net gap space is almost zero. When $N = 200$, in 28/34 datasets, it is hard to distinguish the blue and black curves of Fig. 6. That is, we conclude that using more than a small sample size (like $N = 200$) would not improve defect prediction.

8 Explaining Results of Size Limits for Effective Transfer Learning

To assess the generality of the results in Section 7, we need some background knowledge that knows when a few samples will (or will not) be sufficient to build a defect predictor. Using some sampling theory, this section:

4. In a flock of sheep, the "bellwether" is the individual that the rest of the flock will follow.

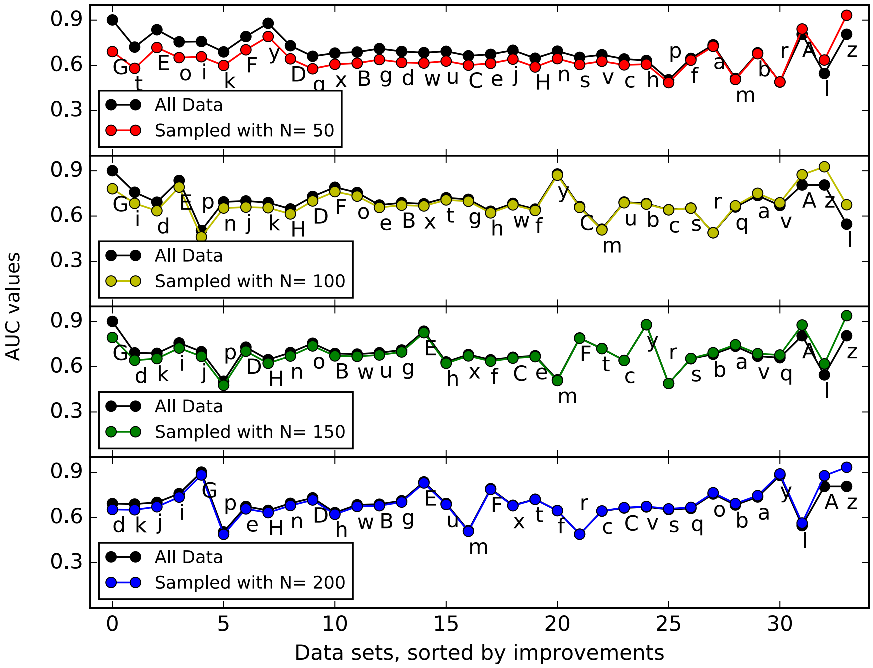


Fig. 6. Improvements of using sampled data over all data with sampled size N = {50, 100, 150, 200}. The right side shows using small dataset is better than using all data. We label the data in table 1 from a to z, and continue from A to H, the last two datasets ar5 and ar6 as G and H.

- Builds such a mathematical model;
- Maps known aspects of defect data into that model;
- Identifies what need to change before the above results no longer hold.

To start, we repeat the *lessons learned* above as well as what is *known about defect datasets*. Next, we define a *maths model* which will be used in a *Monte Carlo simulation* to generate a log of how many samples are required to find some signal. This log will be summarized via a *decision tree learner*.

8.1 Set Up

8.1.1 Lessons from the Above Work

The results in Section 7 show that a small number of examples are sufficient to build a defect predictor, even when the data is transferred from columns with other names. In the following we will build a model to compute the probability that n training examples are sufficient to detect $e\%$ defective instances.

In order to simplify the analysis, we divide n into $n < 50, n < 100, n < 200$, and $n \geq 200$ four ranges respectively (and note that $n \geq 200$ is where the above results do not hold).

8.1.2 Known Aspects About Defect Datasets

Recent results [55], [66] show that, for defect data, good predictors can be built via a *median chop* of numeric project data; they are divided into $b = 2$ bins, i.e., defective bin and non-defective bin. For example, defective instances that likely have high metric values belong to the defective bin while non-defective ones that have low metric values belong to the non-defective bin [66].

Other results [98] show that defect prediction data containing dozens of attributes, many of which are correlated attributes. Hence, while a dataset may have many dimensions, it only really "needs" a few (and by "need" we mean that adding unnecessary dimensions does not add the accuracy of defect predictors learned from this data).

Feature subset selection algorithms [64] can determine which dimensions are needed, and which can be ignored. When applied to defect data [2], we found that those datasets may only need $d \in \{2, 3\}$ dimensions.

Hence, in the following, we will pay particular attention to the "typical" region of $b = 2, d \leq 3$.

8.1.3 A Mathematical Model

Before writing down some maths, it is useful to start with some intuitions. Accordingly, consider a chess board containing small piles of defects in some cells. Like all chess boards, this one is divided into a grid of b^2 cells (in standard chess, $b = 8$ so the board has 64 cells). Further, some cells of the chess board are blank while other cells are $e\%$ covered with that signal.

If we throw a small pebble at that chess board, then the odds of hitting a defect is $c \times p$ where:

- c is the probability of picking a particular cell;
- p is the probability that, once we arrive at that cell, we will find the signal in that cell.

With a few changes, this chess board model can be used to represent the process of machine learning. For example, instead of a board with two dimensions, data mining works on a "chess board" with d dimensions: i.e., one for all the independent variables collected from a project (which are "needed", as defined as Section 8.1.2).

Also, instead of each dimension being divided into eight (like a chess board), it is common in data mining for SE [99] to divide dimensions according to some *descritization policy* [100]. Discretization converts a numeric variable with infinite range into a smaller number of b bins. Hence, the number of cells in a hyper-dimensional chess board is b^d and the probability of selecting any one cell is

$$c = 1/(b^d) = b^{-d}. \tag{5}$$

Once we arrive at any cells, we will be in a region with e percent errors. What is the probability p that we will find those e errors, given n samples from the training data? According to Voas and Miller [101], if we see something at probability e, then we will miss it at probability $1 - e$. After n attempts, the probability of missing it is $(1 - e)^n$ so the probability of stumbling onto e errors is

$$p(e, n) = 1 - (1 - e)^n. \tag{6}$$

The premise of data mining is that in the data "chess board", some cells contain more of the signal than others. Hence, the distribution of the e errors are "skewed" by some factor k. If $k = 1$, then all the errors are evenly distributed over all cells. But at all other values of k, some cells contain more errors than others, computed as follows:

- R_c is a random number $0 \leq R \leq 1$, selected for each part of the space $c \in C$.
- x_c is the proportion of errors in each part of C. $x_c = R^k_{c \in C}$.
- We normalize x_c to be some ratio $0 \leq x_c \leq 1$ as follows: $X = \sum_{c \in C} x_c$ then $x_c = x_c/X$

If e is the ratio of classes within a software project containing errors, then E is the expected value of selecting a cell *and* that cell containing errors

$$E = \sum_{c \in C} c \times x_c e, \tag{7}$$

where c comes from Equation (5) and e is the ratio of classes in the training set with defects.

Using these equations, we can determine how many training examples n are required before $p(E, n)$, from Equation (6), returns a value more than some reasonable threshold T. To make that determination, we call $p(E, n)$ for increasing values of n until $p \geq T$ (for this paper, we used $T = 67\%$).

For completeness, it should be added that the procedure of the above paragraph is an *upper bound* on the number of examples needed to find a signal since it assumes random sampling of a skewed distribution. In practice, if a data mining algorithm is smart, then it would *increase* the probability of finding the target signal, thus *decreasing* how many samples are required.

8.1.4 Monte Carlo Simulation

The above maths let us define a Monte Carlo simulation to assess the external validity of our results. Within 1,000 times of iterations, we picked k, d, b, e values at random from:

- $k \in \{1, 2, 3, 4, 5\}$;
- $d \in \{3, 4, 5, 6, 7\}$ dimensions;
- $b \in \{2, 3, 4, 5, 6, 7\}$ bins;
- $e \in \{0.1, 0.2, 0.3, 0.4\}$

(These ranges were set using our experience with data mining. For example, our prior work shows in defect prediction datasets with 40 or more dimensions, that good predictors can be built using $d \leq 3$ of those dimensions [2].)

Within 1,000 iterations of Monte Carlo simulation, we increased n until Equation (6) showed p passed our reasonable threshold. Next, we generated examples of what n value was found using k, b, d, e.

```
1   dimensions = (1,2)
2   |   dimensions = 1
3   |   |   e ≤ 0.1
4   |   |   |   bins = (1,2,3) : n  <  50
5   |   |   |   bins > 3       : n  <  100
6   |   |   e > 0.1 : n  <  50
7   |   dimensions > 1
8   |   |   bins = (1,2,3)
9   |   |   |   e = 0.1 : n  <  100
10  |   |   |   e > 0.1 : n  <  50
11  |   |   bins > 3
12  |   |   |   bins = (4,5)
13  |   |   |   |   e = 0.1 : n  <  200
14  |   |   |   |   e > 0.1
15  |   |   |   |   |   e ≤ 0.2
16  |   |   |   |   |   |   bins = 4 : n  <  100
17  |   |   |   |   |   |   bins = 5 : n  <  200
18  |   |   |   |   |   e > 0.2      : n  <  100
19  |   |   |   bins > 5
20  |   |   |   |   e ≤ 0.2 : n  ≥  200
21  |   |   |   |   e > 0.2 : n  <  200
22  dimensions > 2
23  |   bins = (1,2)
24  |   |   dimensions = (3,4,5)
25  |   |   |   dimensions = 3
26  |   |   |   |   e ≤ 0.2 : n  <  200
27  |   |   |   |   e > 0.2 : n  <  50
28  |   |   |   dimensions = (4,5)
29  |   |   |   |   e = 0.1 : n  <  200
30  |   |   |   |   e > 0.1
31  |   |   |   |   |   dimensions = 4 : n  <  100
32  |   |   |   |   |   dimensions = 5
33  |   |   |   |   |   |   e ≤ 0.3 : n  <  200
34  |   |   |   |   |   |   e > 0.3 : n  <  100
35  |   |   dimensions > 5 : n  ≥  200
36  |   bins > 2
37  |   |   dimensions = 3
38  |   |   |   bins = (3,4)
39  |   |   |   |   e ≤ 0.3
40  |   |   |   |   |   bins = 3
41  |   |   |   |   |   |   e = 0.1 : n  ≥  200
42  |   |   |   |   |   |   e > 0.1 : n  <  200
43  |   |   |   |   |   bins = 4 : n  ≥  200
44  |   |   |   |   e > 0.3 : n  <  200
45  |   |   |   bins > 4 : n  ≥  200
46  |   |   dimensions > 3 : n  ≥  200
```

Fig. 7. How many n examples are required to be at least 67 percent likely to find defects occurring at probability e.

8.1.5 Decision Tree Learning

These examples were given to a decision tree learner to determine what n values are selected by different ranges of $\{k, b, d, e\}$. Decision tree learners seek an attribute range that, when used to split the data, simplifies the distribution of the dependent variable in each split. The decision tree learner is then called recursively on each split. To test the stability of the learned model, the learning is repeated ten times, each time using 90 percent of the data from training and the rest for testing. The weighted average performance values for the learned decision tree were remarkably good:

- False alarm rates = 2 percent;
- F-measures (i.e., the harmonic mean of recall and precision) of 95 percent

8.2 Results

The resulting decision tree, shown in Fig. 7, defined regions where building defect predictors would be very easy and much harder. Such trees can be read as nested if-then-else statements. For example, Line 1 is an "if", lines 2 to 21 are the associated "then" and the tree starting at Line 22 is the "else". For another example, we could summarise lines 1 to 5 as follows:

> If there are one dimension and the probability of the defects is less than 10 percent then (if the number of bins per dimension is three or less then 50 samples will suffice; else, up to 100 samples may be required.)

In that tree:

- Lines 2 to 6 discuss a very easy case. Here, we only need one dimension to build defect predictors and, for such simple datasets, a few examples are enough for defect prediction.
- Lines 22, 36, 46 show a branch of the decision tree where we need many dimensions that divide into many bins. For such datasets, we require a larger number of samples to learn a predictor ($n \geq 200$).

The key part of Fig. 7 is the "typical" region defined in Section 8.1.2; i.e., $b = 2, d \leq 3$:

- Lines 7 to 10 show one set of branches covering this "typical" region. Note lines 9, 10: we need up to 100 examples when the defect signal is rare (10 percent) but far fewer when the signal occurs at $e > 10$ percent.
- Lines 22 to 27 show another set of branches in this "typical region". Note lines 26, 27: we need up 50 to 200 examples.

8.3 Summary

Our experiments with transfer learning showed that 50 to 200 examples are needed for adequate transfer of defect knowledge. If the reader doubts that this number is too small to be effective, we note that the maths of Section 8 show that this "a small number of examples are enough" is a feature of the kinds of data currently being explored in the defect prediction literature.

9 Discussion

9.1 Practical Guidelines for HDP

We proposed the HDP models to enable defect prediction on software projects by using training datasets from other projects even with heterogeneous metric sets. When we have training datasets in the same project or in other projects with the same metric set, we can simply conduct WPDP or CPDP using recently proposed CPDP techniques respectively [8], [10], [46], [47], [48], [49]. However, in practice, it might be that no training datasets for both WPDP and CPDP exist. In this case, we can apply the HDP approach.

In Section 6 and Table 7, we confirm that many target predictions in HDP by KSAnalyzer with the cutoff of 0.05 outperform or are comparable to baselines and the HDP predictions show 100 percent target coverage. Since KSAnalyzer can match similar source and target metrics, we guide the use of KSAnalyzer for HDP. In terms of the matching score cutoff threshold, there is a trade-off between prediction performance and target coverage. Since a cutoff of 0.05 that is the widely used level of statistical significance [91], we can conduct HDP using KSAnalyzer with the cutoff of 0.05. However, we observe some Loss results in our empirical study. To minimize the percentage of Loss results, we can sacrifice the target coverage by increasing the cutoff as Table 7 shows KSAnalyzer with the cutoff of 0.90 led to 77.8, 100, and 88.9 percent Win results in feasible predictions against CPDP-CM, CPDP-IFS and UDP. By controlling a cutoff value, we may increase the target coverage. For example, we can start from a higher cutoff value and decrease the cutoff until HDP is eligible. This greedy approach might be helpful for practitioners who want to increase the target coverage when conducting HDP. We remain validating this idea as a future work.

9.2 Threats to Validity

We evaluated our HDP models in AUC. AUC is known as a good measure for comparing different prediction models [11], [78], [79], [84]. However, validating prediction models in terms of both precision and recall is also required in practice. To fairly compare WPDP and HDP models in precision and recall, we need to identify a proper threshold of prediction probability. Identifying the proper threshold is a challenging issue and remains as future work.

For RQ1, we computed matching scores using all source and target instances for each prediction combination. With that matching scores, we tested prediction models on a test set from the two-fold cross validation because of the WPDP models as explained in Section 5.4. To conduct WPDP with all instances of a project dataset as a test set, we need a training dataset from the previous releases of the same project. However, the training dataset is not available for our subjects. This may lead to an issue on construct validity since the matching score computations are not based on actual target instances used in the samples of the two-fold cross validation. To address this issue, we additionally conducted experiments with different sample sizes, i.e., 50, 100, 150, and 200 rather using all instances when computing matching scores for HDP in Section 7.

A recent study by Tantithamthavorn et al. [83] pointed out model validation techniques may lead to different interpretation of defect prediction results. Although the n-fold cross validation is one of widely used model validation techniques [8], [80], [81], our experimental results based on the two-fold cross validation may be different from those using other validation techniques. This could be an issue in terms of construct validity as well.

Since we used the default options for machine learners in our experiments, the experimental results could be improved further when we use optimized options [102], [103]. Thus, our results may be affected by the other options tuning machine learners. We remain conducting experiments with the optimized options as a future work.

10 Conclusion

In the past, cross-project defect prediction cannot be conducted across projects with heterogeneous metric sets. To address this limitation, we proposed heterogeneous defect prediction based on metric matching using statistical analysis [67]. Our experiments showed that the proposed HDP models are feasible and yield promising results. In addition, we investigated the lower bounds of the size of source and target datasets for effective transfer learning in defect prediction. Based on our empirical and mathematical studies, can show categories of data sets were as few as 50 instances are enough to build a defect predictor and apply HDP.

HDP is very promising as it permits potentially all heterogeneous datasets of software projects to be used for defect prediction on new projects or projects lacking in defect data. In addition, it may not be limited to defect prediction. This technique can potentially be applicable to all prediction and recommendation based approaches for software engineering problems. As future work, for the metric matching, we will apply other techniques, like deep learning, to explore new features from source and target projects to improve the performance. Since transfer learning has shown such great power, we will explore the feasibility of building various prediction and recommendation models to solve other software engineering problems.

REFERENCES

[1] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empirical Softw. Eng.*, vol. 17, no. 4/5, pp. 531–577, 2012.

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

[3] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Softw. Eng.*, vol. 22, no. 10, pp. 751–761, Oct. 1996.

[4] M. H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*. New York, NY, USA: Elsevier, 1977.

[5] T. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.

[6] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 432–441.

[7] T. Lee, J. Nam, D. Han, S. Kim, and I. P. Hoh, "Micro interaction metrics for defect prediction," in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2011, pp. 311–321.

[8] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 382–391.

[9] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Softw. Eng.*, vol. 19, no. 2, pp. 167–199, 2012.

[10] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Inf. Softw. Technol.*, vol. 54, no. 3, pp. 248–256, Mar. 2012.

[11] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the "imprecision" of cross-project defect prediction," presented at the *ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng.*, New York, NY, USA, 2012.

[12] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Softw. Eng.*, vol. 14, pp. 540–578, Oct. 2009.

[13] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: A large scale experiment on data versus domain versus process," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, 2009, pp. 91–100.

[14] T. Menzies, et al., "The promise repository of empirical software engineering data," Jun. 2012. [Online]. Available: http://promisedata.googlecode.com

[15] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: Examining the effects of ownership on software quality," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng.*, 2011, pp. 4–14. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025119

[16] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *IEEE Trans. Softw. Eng.*, vol. 22, no. 12, pp. 886–894, Dec. 1996. [Online]. Available: http://dx.doi.org/10.1109/32.553637

[17] J. Nam and S. Kim, "Heterogeneous defect prediction," in *Proc. 10th Joint Meet. Found. Softw. Eng.*, 2015, pp. 508–519. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786814

[18] IEEE-1012, *IEEE Standard 1012–2004 for Software Verification and Validation, IEEE Standard 1012–2004*, 1998.

[19] A. Endres and D. Rombach, *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories*. Reading, MA, USA: Addison Wesley, 2003.

[20] C. Jones, *Software Engineering Best Practices*, 1st ed. Boston, MA, USA: McGraw Hill, 2010.

[21] R. L. Glass, *Facts and Fallacies of Software Engineering*. Boston, MA, USA: Addison-Wesley, 2002.

[22] D. Budgen, "Why should they believe us? Determinism, non-determinism and evidence" *19th Conf. Softw. Eng. Educ. Training (CSEET'06)*. p. 4, Apr. 2006, doi: 10.1109/CSEET.2006.41.

[23] B. K. David Budgen and Pearl Brereton, "Is evidence based software engineering mature enough for practice & policy?" presented at the *33rd Annu. IEEE Softw. Eng. Workshop*, Skovde, Sweden, 2009.

[24] T. Menzies, et al., "Local versus global lessons for defect prediction and effort estimation," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 822–834, Jun. 2013.

[25] N. Bettenburg, M. Nagappan, and A. E. Hassan, "Towards improving statistical modeling of software engineering data: Think locally, act globally!" *Empirical Softw. Eng.*, pp. 1–42, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10664–013-9292-6

[26] D. Posnett, V. Filkov, and P. Devanbu, "Ecological inference in empirical software engineering," in *Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2011, pp. 362–371. [Online]. Available: http://dx.doi.org/10.1109/ASE.2011.6100074

[27] Y. Yang, et al., "Local bias and its impacts on the performance of parametric estimation models," in *Proc. 7th Int. Conf. Predictive Models Softw. Eng.*, 2011, Art. no. 14.

[28] R. Krishna, T. Menzies, and W. Fu, "Too much automation? the bellwether effect and its implications for transfer learning," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2016, pp. 122–131. [Online]. Available: http://doi.acm.org/10.1145/2970276.2970339

[29] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *Proc. 27th Int. Conf. Softw. Eng.*, 2005, pp. 580–586.

[30] T. Menzies, D. Raffo, S. Setamanit, Y. Hu, and S. Tootoonian, "Model-based tests of truisms," in *Proc. 17th IEEE Int. Conf. Automated Softw. Eng.*, 2002, Art. no. 183.

[31] C. Lewis, Z. Lin, C. Sadowski, X. Zhu, R. Ou, and E. J. W. Jr, "Does bug prediction support human developers? Findings from a Google case study," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 372–381.

[32] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proc. 28th Int. Conf. Softw. Eng.*, 2006, pp. 452–461. [Online]. Available: http://doi.acm.org/10.1145/1134285.1134349

[33] T. Ostrand, E. Weyuker, and R. Bell, "Predicting the location and number of faults in large software systems," *IEEE Trans. Softw. Eng.*, vol. 31, no. 4, pp. 340–355, Apr. 2005.

[34] M. Kim, J. Nam, J. Yeon, S. Choi, and S. Kim, "REMI: Defect prediction for efficient API testing," in *Proc. 10th Joint Meet. Found. Softw. Eng.*, 2015, pp. 990–993. [Online]. Available: http://doi.acm.org/10.1145/2786805.2804429

[35] S. Rakitin, *Software Verification and Validation for Practitioners and Managers*, 2nd ed. Norwood, MA, USA: Artech House, 2001.

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

[37] A. Tosun, A. Bener, and R. Kale, "AI-based software defect predictors: Applications and benefits in a case study," in *Proc. 22nd Conf. Innovative Appl. Artif. Intell.*, 2010, pp. 57–68.

[38] A. Tosun, A. Bener, and B. Turhan, "Practical considerations of deploying ai in defect prediction: A case study within the Turkish telecommunication industry," in *Proc. 5th Int. Conf. Predictive Models Softw. Eng.*, 2009, Art. no. 11.

[39] F. Shull et al., "What we have learned about fighting defects," in *Proc. 8th Int. Softw. Metrics Symp.*, 2002, pp. 249–258.

[40] M. Fagan, "Design and code inspections to reduce errors in program development," *IBM Syst. J.*, vol. 15, no. 3, pp. 182–211, 1976.

[41] F. Rahman, S. Khatri, E. Barr, and P. Devanbu, "Comparing static bug finders and statistical prediction," in *Proc. Int. Conf. Softw. Eng.*, 2014, pp. 424–434. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568269

[42] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[43] B. Sigweni and M. Shepperd, "Feature weighting techniques for CBR in software effort estimation studies: A review and empirical evaluation," in *Proc. 10th Int. Conf. Predictive Models Softw. Eng.*, 2014, pp. 32–41.

[44] V. Dallmeier, N. Knopp, C. Mallon, G. Fraser, S. Hack, and A. Zeller, "Automatically generating test cases for specification mining," *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 243–257, Mar./Apr. 2012.

[45] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *J. Big Data*, vol. 3, no. 1, pp. 1–40, 2016.

[46] G. Canfora, A. De Lucia , M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Multi-objective cross-project defect prediction," in *Proc. IEEE 6th Int. Conf. Softw. Testing Verification Validation*, Mar. 2013, pp. 252–261.

[47] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'Union fait la force," in *Proc. IEEE Conf. Softw. Evol. Week Softw. Maintenance Reengineering Reverse Eng.*, Feb. 2014, pp. 164–173.

[48] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," *Empirical Softw. Eng.*, pp. 1–29, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10664–014-9346-4

[49] D. Ryu, J.-I. Jang, and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction," *Softw. Quality J.*, pp. 1–38, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11219–015-9287-1

[50] Y. Zhang, D. Lo, X. Xia, and J. Sun, "An empirical study of classifier combination for cross-project defect prediction," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, Jul. 2015, vol. 2, pp. 264–269.

[51] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter language reuse," in *Proc. 4th Int. Workshop Predictor Models Softw. Eng.*, 2008, pp. 19–24.

[52] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in *Proc. 11th Working Conf. Mining Softw. Repositories*, 2014, pp. 172–181.

[53] Y. Kamei, et al., "A large-scale empirical study of just-in-time quality assurance," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 757–773, Jun. 2013.

[54] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Inf. Softw. Technol.*, vol. 59, pp. 170–190, 2015.

[55] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," in *Proc. 11th Working Conf. Mining Softw. Repositories*, 2014, pp. 182–191.

[56] P. He, B. Li, and Y. Ma, "Towards cross-project defect prediction with imbalanced feature sets," *CoRR*, 2014. [Online]. Available: http://arxiv.org/abs/1411.4228

[57] X. Jing, F. Wu, X. Dong, F. Qi, and B. Xu, "Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, 2015, pp. 496–507. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786813

[58] E. Shihab, "Practical software quality prediction," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Sep. 2014, pp. 639–644.

[59] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, Mar. 2003.

[60] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: An investigation on feature selection techniques," *Softw. Practice Experience*, vol. 41, no. 5, pp. 579–606, Apr. 2011. [Online]. Available: http://dx.doi.org/10.1002/spe.1043

[61] S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim, "Reducing features to improve code change-based bug prediction," *IEEE Trans. Softw. Eng.*, vol. 39, no. 4, pp. 552–569, Apr. 2013.

[62] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explorations Newslett.*, vol. 11, pp. 10–18, Nov. 2009.

[63] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Inf. Sci.*, vol. 179, no. 8, pp. 1040–1058, 2009.

[64] M. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 6, pp. 1437–1447, Nov. 2003.

[65] H. Liu, J. Li, and L. Wong, "A comparative study on feature selection and classification methods using gene expression profiles and proteomic patterns," *Genome Informat.*, vol. 13, pp. 51–60, 2002.

[66] J. Nam and S. Kim, "CLAMI: Defect prediction on unlabeled datasets (T)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2015, pp. 452–463. [Online]. Available: http://dx.doi.org/10.1109/ASE.2015.56

[67] F. J. Massey, "The Kolmogorov-Smirnov test for goodness of fit," *J. Amer. Statist. Assoc.*, vol. 46, no. 253, pp. 68–78, 1951.

[68] C. Spearman, "The proof and measurement of association between two things," *Int. J. Epidemiology*, vol. 39, no. 5, pp. 1137–1150, 2010.

[69] J. Matouek and B. Gärtner, *Understanding and Using Linear Programming (Universitext)*. Secaucus, NJ, USA: Springer-Verlag, 2006.

[70] H. W. Lilliefors, "On the Kolmogorov-Smirnov test for normality with mean and variance unknown," *J. Amer. Statist. Assoc.*, vol. 62, no. 318, pp. pp. 399–402, 1967.

[71] R. Wu, H. Zhang, S. Kim, and S. Cheung, "ReLink: Recovering links between bugs and changes," in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2011, pp. 15–25.

[72] F. Peters and T. Menzies, "Privacy and utility for defect prediction: Experiments with MORPH," in *Proc. Int. Conf. Softw. Eng.*, 2012, pp. 189–199.

[73] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect datasets," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, Sep. 2013.

[74] Understand 2.0. [Online]. Available: http://www.scitools.com/products/

[75] X. Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Trans. Softw. Eng.*, vol. 43, no. 4, pp. 321–339, Apr. 2017.

[76] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 309–320. [Online]. Available: http://doi.acm.org/10.1145/2884781.2884839

[77] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, pp. 789–800.

[78] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, Jul./Aug. 2008.

[79] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 356–370, May/Jun. 2011.

[80] M. Kläs, F. Elberzhager, J. Münch, K. Hartjes, and O. von Graevemeyer, "Transparent combination of expert and measurement data for defect prediction: An industrial case study," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, 2010, pp. 119–128.

[81] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?" in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2008, pp. 2–12.

[82] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proc. 33rd Int. Conf. Softw. Eng.*, 2011, pp. 1–10.

[83] C. Tantithamthavorn, S. McIntosh, A. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 43, no. 1, pp. 1–18, Jan. 2017.

[84] E. Giger, M. D'Ambros, M. Pinzger, and H. C. Gall, "Method-level bug prediction," in *Proc. ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2012, pp. 171–180.

[85] T. Mende, "Replication of defect prediction studies: Problems, pitfalls and recommendations," in *Proc. 6th Int. Conf. Predictive Models Softw. Eng.*, 2010, pp. 5:1–5:10.

[86] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's δ for evaluating group differences on the NSSE and other surveys?" in *Proc. Annu. Meeting Florida Assoc. Institutional Res.*, 2006, pp. 1–3.

[87] E. Kocaguneli, T. Menzies, J. Keung, D. Cok, and R. Madachy, "Active learning and effort estimation: Finding the essential content of software effort estimation data," *IEEE Trans. Softw. Eng.*, vol. 39, no. 8, pp. 1040–1053, Aug. 2013.

[88] M. Li, H. Zhang, R. Wu, and Z.-H. Zhou, "Sample-based software defect prediction with active and semi-supervised learning," *Automated Softw. Eng.*, vol. 19, no. 2, pp. 201–230, 2012.

[89] G. Valentini and T. G. Dietterich, "Low bias bagged support vector machines," in *Proc. 20th Int. Conf. Mach. Learning*, 2003, pp. 752–759.

[90] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, Dec. 1945.

[91] G. W. Corder and D. I. Foreman, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. Hoboken, NJ, USA: Wiley, 2009.

[92] D. Kim, Y. Tao, S. Kim, and A. Zeller, "Where should we fix this bug? A two-phase recommendation model," *IEEE Trans. Softw. Eng.*, vol. 39, no. 11, pp. 1597–1610, Nov. 2013.

[93] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994.

[94] B. Turhan, "On the dataset shift problem in software engineering prediction models," *Empirical Softw. Eng.*, vol. 17, no. 1/2, pp. 62–74, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10664-011-9182-8

[95] N. Landwehr, M. Hall, and E. Frank, "Logistic model trees," *Mach. Learn.*, vol. 59, no. 1/2, pp. 161–205, 2005.

[96] J. Bruin, "newtest: Command to compute new test," Feb. 2011. [Online]. Available: http://www.ats.ucla.edu/stat/stata/ado/analysis/

[97] Y. Yang, et al., "Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2016, pp. 157–168. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950353

[98] M. Shepperd and D. C. Ince, "A critique of three metrics," *J. Syst. Softw.*, vol. 26, no. 3, pp. 197–210, Sep. 1994.

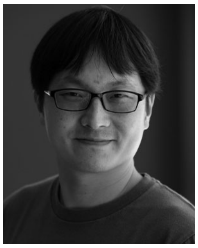
[99] T. Menzies, "Data mining," in *Recommendation Systems in Software Engineering*. Berlin, Germany: Springer, 2014, pp. 39–75.

[100] J. L. Lustgarten, V. Gopalakrishnan, H. Grover, and S. Visweswaran, "Improving classification performance with discretization on biomedical datasets," in *Proc. AMIA Annu. Symp. Proc.*, 2008, pp. 445–449.

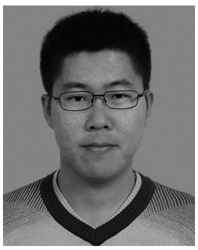
[101] J. M. Voas and K. W. Miller, "Software testability: The new verification," *IEEE Softw.*, vol. 12, no. 3, pp. 17–28, May 1995.

[102] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 321–332. [Online]. Available: http://doi.acm.org/10.1145/2884781.2884857

[103] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?" *Inf. Softw. Technol.*, vol. 76, pp. 135–146, 2016.



Jaechang Nam received the BEng from Handong Global University, Korea, in 2002, the MSc in computer science from Blekinge Institute of Technology, Sweden, in 2009, and the PhD degree in computer science and engineering from the Hong Kong University of Science and Technology in 2015. He was a postdoctoral fellow at University of Waterloo, Canada, from Sep. 2015 to Sep. 2017 and a research assistant professor at POSTECH, Korea, from Oct. 2017 to Feb. 2018. Since Mar. 2018, he has been an assistant professor at Handong Global University where he received his BEng. His research interests include software quality, mining software repositories, and empirical software engineering.



Wei Fu is currently working toward the PhD degree in computer science at North Carolina State University. His research interests include search-based software engineering, software repository mining, and artificial intelligence applications to improve software quality. He is a student member of the IEEE and the ACM.

Sunghun Kim received the PhD degree from the Department of Computer Science, University of California, Santa Cruz, in 2006. He is an associate professor of computer science with the Hong Kong University of Science and Technology. He was a postdoctoral associate in the Massachusetts Institute of Technology and a member of the Program Analysis Group. He was the chief technical officer (CTO) and led a 25-person team for six years at the Nara Vision Co. Ltd., a leading Internet software company in Korea. His core research area is software engineering, focusing on software evolution, program analysis, and empirical studies. He is a member of the IEEE.

Tim Menzies received the PhD degree from the UNSW, in 1995. He is a full professor in CS at North Carolina State University, where he teaches SE, programming languages and foundations of software science. He is the director of the RAISE lab (real world AI for SE) that explores SE, data mining, AI, search-based SE, and open access science. An author of more than 250 referred publications, he has been a lead researcher on projects for US National Science Foundation, NIJ, DoD, NASA, USDA, as well as joint research work with private companies. For 2002 to 2004, he was the software engineering research chair at NASA's software Independent Verification and Validation Facility. He is the co-founder of the PROMISE conference series devoted to reproducible experiments in software engineering (http://tiny.cc/seacraft). He is an associate editor of the *IEEE Transactions on Software Engineering*, the *Empirical Software Engineering*, the *Automated Software Engineering Journal*, the *Big Data Journal*, the *Information Software Technology*, and the *Software Quality Journal*. He has served as the co-PC chairs for ASE'12, ICSE'15 NIER, SSBSE'17, and the co-general chair of ICMSE'16. He is a member of the IEEE.

Lin Tan received the PhD degree from the University of Illinois, Urbana-Champaign. She is the Canada research chair in software dependability and an associate professor in the Department of Electrical and Computer Engineering, University of Waterloo. She is on the editorial board of the Springer *Empirical Software Engineering Journal* (2015-present). She was the program co-chairs of MSR 2017, ICSE-NIER 2017, and ICSME-ERA 2015. She is a recipient of an NSERC Discovery Accelerator Supplements Award, an Ontario Early Researcher Award, an Ontario Professional Engineers Award—Engineering Medal for Young Engineer, a University of Waterloo Outstanding Performance Award, two Google Faculty Research Awards, and an IBM CAS Research Project of the Year Award. Her coauthored papers have received an ACM SIGSOFT Distinguished Paper Award at FSE in 2016 and IEEE Micro's Top Picks in 2006. She is a member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.