Heterogeneous Defect Prediction

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

Abstract—Much recent work has 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, much of that work 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. HDP's metric matching 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 as small as 200 instances (with as low as 20 defective instances). That is, 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

Most defect prediction models are based on machine learning. Therefore, it is a must to collect defect datasets to train a prediction model [1], [2]. The defect datasets consist of various software metrics and labels. Software metrics are the terminology describing software projects. Commonly used software metrics for defect prediction are complexity metrics (such as lines of code, Halstead metrics, McCabe's cyclomatic complexity, and CK metrics) and process metrics [3], [4], [5], [6]. 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]. In Figure 1a, 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.

However, it is difficult to build a prediction model for new software projects or projects with little historical information [9] since they do not have enough training instances. Various process metrics and label information can be extracted from the historical data of software repositories such as version control and issue tracking systems [6]. Thus, it is difficult to collect process metrics and instance labels in new projects or projects that have little historical data [8], [9], [10]. For example, without instances being labeled using past defect data, it is not possible to build a prediction model.

To address this issue, researchers have proposed cross-project defect prediction (CPDP) [8], [9], [11], [12], [13], [14]. CPDP approaches predict defects even for new projects lacking in historical data by reusing prediction models built by other project datasets. As shown in Figure 1b, a prediction model is trained by labeled instances in *Project A* (source) and predicts defects in *Project B* (target).

However, most CPDP approaches have a serious limitation: CPDP is only feasible for projects which have exactly the same metric set as shown in Figure 1b. Finding other projects with exactly the same metric set can be challenging. Publicly available defect datasets that are widely used in defect prediction literature usually have heterogeneous metric sets [1], [8], [15]. For example, many NASA datasets in the PROMISE repository have 37 metrics but AEEEM datasets used by D'Ambroas et al. have 61 metrics [1], [15]. 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 [14].

Some CPDP studies use only common metrics when source and target datasets have heterogeneous metric sets [12], [14]. For example, Turhan et al. use the only 17 common metrics between the NASA and SOFTLAB datasets that have heterogeneous metric sets [14]. However, 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 vs. 0.39) [14].

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 Figure 1c, we could reuse any

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

Manuscript received January XX, 2016; revised April XX, 2016.

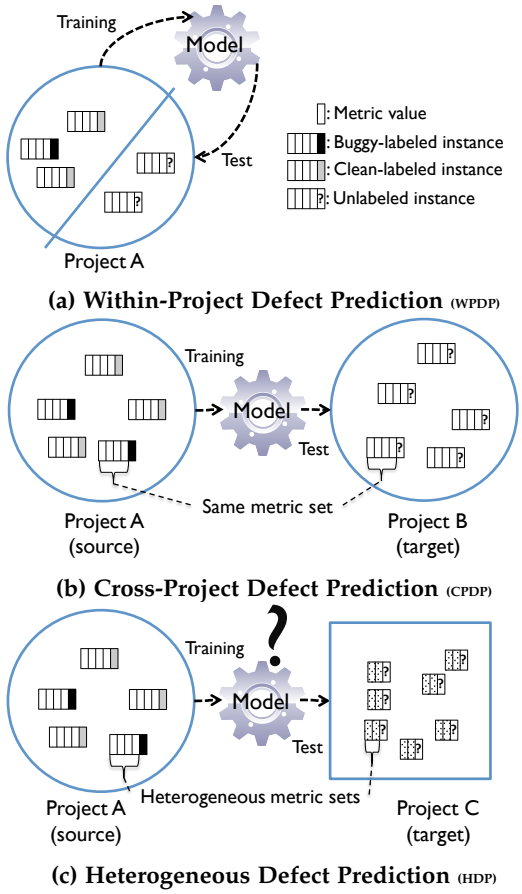

Fig. 1: Various Defect Prediction Scenarios

existing defect datasets to build a prediction model. For example, many PROMISE defect datasets even if they have heterogeneous metric sets [15] could be used as training datasets to predict defects in any project.

The key idea of our HDP approach is to transfer knowledge from a source dataset to predict defects in a target dataset by matching metrics that have similar distributions between source and target datasets. 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 to build a prediction model. We address this limit in this paper as well.

1.1 Research Questions

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

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

1.2 Contributions

Our experimental study shows that HDP models are feasible and their prediction performance is promising. About 68% of HDP predictions are better or comparable to WPDP predictions with statistical significance.

For RQ2, we conducted the experimental study by using various sampling sizes of source and target datasets for HDP and validate the generality of its results through a *Monte Carlo* simulation. Our results suggest that 200 instances for source (with at least 20 defective samples) and target datasets could be effective enough for our HDP models.

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.
- Validating the lower bounds of the size of source and target datasets for effective heterogeneous defect prediction.

1.3 Extensions from Prior Publication

We extend the previous conference paper of the same name [16] in the following ways. First, we motivate this study in the view of transfer learning in 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 Section 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, 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 [17] proposes numerous methods for assessing software quality;
- Endres & Rombach catalog dozens of lessons of software engineering [18] 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 [19] and Glass [20] who list (for exmple) Brooks' Law (adding programmers to a late project makes it later).
- More generally, Budgen & Kitchenham seek to reorganize SE research using general conclusions drawn from a larger number of studies [21], [22].

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. Posnett et al. [23] 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, files) and show that models that work at one level of aggregation can be sub-optimal at others. For example, Yang et al. [24], Bettenburg et al. [25], and Menzies et al. [26] all explore the generation of models using *all* data versus *local* samples that 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:

- Finding the reference sets (this paper shows that finding them may not be a too complex task, at least for defect prediction).
- 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 point #1.
- Transferring lessons from the reference set to new projects.

In this approach, the policies of the organization will be 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 point #3: transferring lessons from the reference set to new projects. To support this third point, we need to resolve the problems that this paper addresses (data expressed in different terminology cannot transfer till there is enough data to match old projects to new).

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 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 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 static code attributes. Given static code descriptors for each module, plus a count of humber 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: Static code attributes can be automatically collected, even for very large systems [27]. 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 [28].

Widely used: Researchers and industrial practitioners use static attributes to guide software quality predictions. Defect prediction models have been reported at Google [29]. Verification and validation (V&V) textbooks [30] advise using static code complexity attributes to decide which modules are worth manual inspections.

Useful: Defect predictors often find the location of 70% (or more) of the defects in code [31]. Defect predictors have some level of generality: predictors learned at NASA [31] have also been found useful elsewhere (e.g. in Turkey [32], [33]). 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* [34] concluded that manual software reviews can find $\approx 60\%$ of defects. In another work, Raffo documents the typical defect detection capability of industrial review methods: around 50% for full Fagan inspections [35] to 21% for less-structured inspections.

Not only do static code 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. [36] compared (a) static code analysis tools FindBugs, Jlint, and Pmd and (b) static code defect predic-

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.

tors (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 static code defect prediction can be quickly adapted to new languages by building lightweight parsers to extract high-level static code features. 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 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], [9], [12], [13], [14], [37], [38], [39], [40], [41]. Since the performance of CPDP is usually very poor [9], researchers have proposed various techniques to improve CPDP [8], [12], [14], [37], [38], [39], [40], [42]. In this section, we discuss CPDP studies in terms of metric sets in defect prediction datasets.

3.1 CPDP Using Same/common Metric Sets

Watanabe et al. proposed the metric compensation approach for CPDP [42]. The metric compensation transforms a target dataset similar to a source dataset by using the average metric values [42]. 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 [42].

Rahman et al. evaluated the CPDP performance in terms of cost-effectiveness and confirmed that the prediction performance of CPDP is comparable to WPDP [13]. For the empirical study, Rahman et al. collected 9 datasets with the same process metric set [13].

Fukushima et al. conducted an empirical study of just-in-time defect prediction in the CPDP setting [10]. They used 16 datasets with the same metric set [10]. 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 [10], [43].

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 [14]. The basic idea of the NN filter is that prediction models are built by source instances that are nearest-neighbours of target instances [14]. To conduct CPDP, Turhan et al. used 10 NASA and SOFTLAB datasets in the PROMISE repository [14], [15].

Ma et al. proposed Transfer Naive Bayes (TNB) [12]. The TNB builds a prediction model by weighting source instances similar to target instances [12]. Using the same datasets used by Turhan et al., Ma et al. evaluated the TNB models for CPDP [12], [14].

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 [12], [14]. There is another CPDP study with the top-K common metric subset [44]. However, as explained in Section 1, CPDP using common metrics is worse than WPDP [14], [44].

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% of defect datasets in the PROMISE repository have the same metric set [15]. In other words, we cannot directly conduct CPDP for the 18% of the defect datasets by using the remaining (82%) datasets in the PROMISE repository [15].

There are other CPDP studies using datasets with the same metric sets or using common metric sets [15], [26], [37], [38], [39], [40], [41]. Menzies et al. proposed a local prediction model based on clustering [26]. They used seven defect datasets with 20 object-oriented metrics from the PROMISE repository [15], [26]. Canfora et al., Panichella et al., and Zhang et al. used ten Java projects only with the same metric set from the PROMISE repository [15], [37], [38], [41]. Ryu et al. proposed the value-cognitive boosting and transfer cost-sensitive boosting approaches for CPDP [39], [40]. Ryu et al. used common metrics in NASA and SOFTLAB datasets [39] or Jureczko datasets with the same metric set from the PROMISE repository [40]. 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 [45]. The universal model is built using 1398 projects from SourceForge and Google code and leads to comparable prediction results to WPDP in their experimental setting [45].

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 CPDP Using Heterogeneous Metric Sets

He et al. [46] 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 [46].

However, the approach by He et al. is not compared with WPDP [46]. Although their best approach is helpful to improve regular CPDP models, the evaluation might be weak since the prediction performance of a regular CPDP

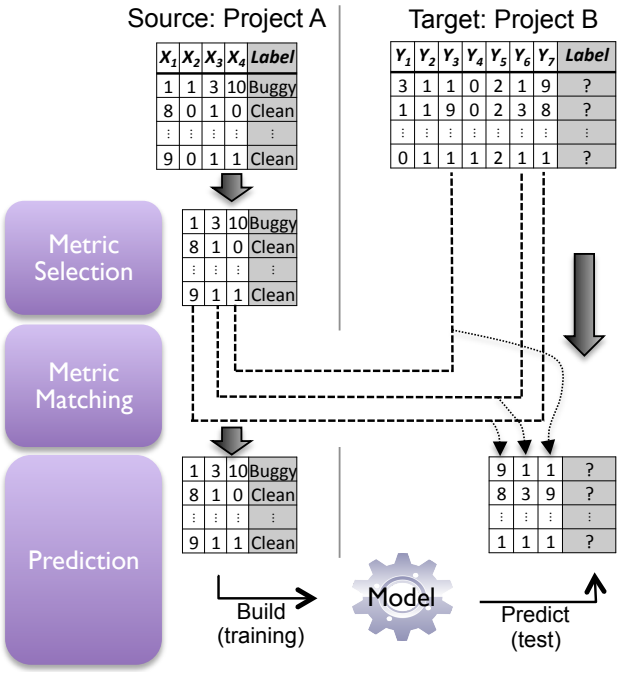

Fig. 2: Heterogeneous defect prediction

is usually very poor [9]. In addition, He et al. conducted experiments on only 11 projects in 3 dataset groups [46].

Jing et al. proposed heterogeneous cross-company defect prediction based on the extended canonical correlation analysis (CCA+) [47] 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. [46], we compare HDP to WPDP, and HDP achieved better or comparable prediction performance to WPDP in about 68% of predictions. Comparing to the experiments for CCA+ [47] with 14 projects, we conducted more extensive experiments with 28 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+ [47]. 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 [16], [48]. In Section 4, we describe our approach in detail.

4 Approach

Figure 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).

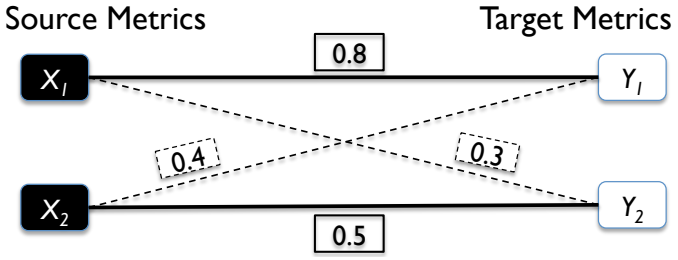

Fig. 3: An example of metric matching between source and target datasets.

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 [49]. We apply widely used feature selection techniques for metric selection of a source dataset as in Section 4.1 [50], [51].

After that, metrics based on their similarity such as distribution or correlation between the source and target metrics are matched up. In Figure 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 subsections, 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 [50], [51]. According to benchmark studies about various feature selection approaches, a single best feature selection approach for all prediction models does not exist [52], [53], [54]. For this reason, we conduct experiments under different feature selection approaches. When applying feature selection approaches, we select top 15% of metrics as suggested by Gao et al. [50]. In addition, we compare the prediction results with or without metric selection in the experiments.

4.2 Matching Source and Target Metrics

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 [55], [56]. We define the following three analyzers for metric matching:

- 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. Figure 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 Figure 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 Figure 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 Figure 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* [57] technique to select a group of matched metrics, whose sum of matching scores is highest, without duplicated metrics. In Figure 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 subsections.

4.2.1 PAnalyzer

PAnalyzer simply compares 9 percentiles (10th, 20th,..., 90th) of ordered values between source and target metrics.

First, we compute the difference of n-th 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 n-th percentiles of i-th source and j-th target metrics, and $sp_{ij}(n)$ and $bp_{ij}(n)$ are smaller and bigger percentile values respectively at n-th percentiles of i-th source and j-th 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 ($P_{ij}(10) = 15/20 = 0.75$).

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

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

, where M_{ij} is a matching score between i-th source and j-th target metrics. 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.

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 two sample test that can be applicable when we cannot be sure about the normality of two samples and/or the same variance [55], [58]. 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 compute p-values for these datasets. In statistical testing, a p-value shows the probability of whether two samples are significantly different or not. We used the *KolmogorovSmirnovTest* implemented in the *Apache commons math* library.

The matching score is:

$$M_{ij} = p_{ij} \tag{3}$$

, where p_{ij} is a p-value from the KS-test of i-th source and j-th target metrics. A p-value tends to be zero when two metrics are significantly different.

4.2.3 SCoAnalyzer

In SCoAnalyzer, we used the Spearman's rank correlation coefficient as a matching score for source and target metrics [56]. Spearman's rank correlation measures how two samples are correlated [56]. To compute the coefficient, we used the *SpearmansCorrelation* in the *Apache commons math* 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:

$$M_{ij} = c_{ij} \tag{4}$$

, where c_{ij} is a Spearman's rank correlation coefficient between i-th source and j-th 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.

TABLE 1: The 28 defect datasets from five groups.

Group	Dataset	# of instances		# of metrics	Prediction Granularity
		All	Buggy(%)		
AEEEM [1], [8]	EQ	325	129(39.69%)	61	Class
	JDT	997	206(20.66%)		
	LC	399	64(9.26%)		
	ML	1862	245(13.16%)		
	PDE	1492	209(14.01%)		
ReLink [59]	Apache	194	98(50.52%)	26	File
	Safe	56	22(39.29%)		
	ZXing	399	118(29.57%)		
MORPH [60]	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 [15], [61]	cm1	327	42(12.84%)	37	Function
	mw1	253	27(10.67%)		
	pc1	705	61(8.65%)		
	pc3	1077	134(12.44%)		
	pc4	1458	178(12.21%)		
SOFTLAB [14]	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%)		

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], [14], [59], [60]. 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 28 defect datasets: AEEEM, ReLink, MORPH, NASA, and SOFTLAB.

AEEEM was used to benchmark different defect prediction models [1] and to evaluate CPDP techniques [8], [46]. 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. [59] 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 [62].

The MORPH group contains defect datasets of several open source projects used in the study about the dataset privacy issue for defect prediction [60]. The 20 metrics used in MORPH are McCabe's cyclomatic metrics, CK metrics, and other OO metrics [60].

NASA and SOFTLAB contain proprietary datasets from NASA and a Turkish software company, respectively [14]. We used five NASA datasets, which share the same metric set in the PROMISE repository [15], [61]. We used cleaned NASA datasets (DS′ version) [61]. For the SOFTLAB group, we used all SOFTLAB datasets in the PROMISE repository [15]. 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* [15].

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]

We did not conduct defect prediction across projects in the same group 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 600 possible prediction combinations from these 28 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 ≤ 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 on average.

5.3 Baselines

We compare HDP to three baselines: WPDP (Baseline1), CPDP using common metrics (CPDP-CM) between source and target datasets (Baseline2), and CPDP-IFS (Baseline3).

We first compare HDP to WPDP. Comparing HDP to WPDP will provide empirical evidence of whether our HDP models are applicable in practice.

We conduct CPDP using only common metrics (CPDP-CM) between source and target datasets as in previous CPDP studies [12], [14], [46]. For example, AEEEM and MORPH have OO metrics as common metrics so we use them to build prediction models for datasets between AEEEM and MORPH. Since using common metrics has been adopted to address the limitation on heterogeneous metric sets in previous CPDP studies [12], [14], [46], we set CPDP-CM as a baseline to evaluate our HDP models. The number of matched metrics varies across the dataset group. Between AEEEM and ReLink, only one common metric exists, *LOC*.

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

NASA and SOFTLAB have 28 common metrics. On average, the number of common metrics in our datasets are about five.

We include CPDP-IFS proposed by He et al. as a baseline [46]. 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 such as 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 [46]. The 16 distribution characteristics are used as features to build a prediction model [46].

5.4 Experimental Design

For the machine learning algorithm, we use Logistic Regression, which is widely used for both WPDP and CPDP studies [8], [9], [63], [64]. We use Logistic Regression implemented in Weka with default options [65].

For WPDP, it is necessary to split datasets into training and test sets. We use 50:50 random splits, which are widely used in the evaluation of defect prediction models [8], [66], [67]. For the 50:50 random splits, we use one half of the instances for training a model and the rest for test (round 1). In addition, we use the two splits 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. 1000 tests, since there is a randomness in selecting instances for each split [68]. Simply speaking, we repeat the two-fold cross validation 500 times.

For CPDP-CM, CPDP-IFS, and HDP, we build a prediction model by using a source dataset and test the model on the same test splits used in WPDP. Since there are 1000 different test splits for a within-project prediction, the CPDP-CM, CPDP-IFS, 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). 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 [13], [69], [70], [71]. 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 [72]. 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 [13].

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 [73]. 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 ($0.474 \leq |\delta|$) [73].

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 [74], [75], [76]. As we repeat the experiments 1000 times for a target project dataset, we conduct the Wilcoxon signed-rank test ($p<0.05$) for all AUC values in baselines and HDP [77]. 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 and existing CPDP approaches for heterogeneous metric sets (CPDP-CM and CPDP-IFS)?

RQ1 leads us to investigate whether our HDP is comparable to WPDP (Baseline1), CPDP-CM (Baseline2), and CDDP-IFS (Baseline3) [46]. We report the representative HDP results in Section 6.1, 6.2, and 6.3 based on significance attribute selection for metric selection and KSAnalyzer with the cutoff threshold of 0.05. Among different metric selections, significance attribute selection 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 a commonly accepted significance level in the statistical test [78].

In Section 6.4, 6.5, and 6.6, 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, for each target as well as *all* targets (the second last row in the table). 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. [46]. The last column shows the HDP results by KSAnalyzer with the cutoff of 0.05. If there are better results between Baseline1 and our approach with statistical significance (Wilcoxon signed-rank test [77], $p<0.05$), the better AUC values are in bold font as shown in Table 2. Between Baseline2 and our approach, better AUC values

TABLE 2: Comparison results among WPDP, CPDP-CM, CPDP-IFS, and HDP by KSAnalyzer with the cutoff of 0.05 in a median AUC.

Target	WPDP (Baseline1)	CPDP-CM (Baseline2)	CPDP-IFS (Baseline3)	HDP KSAnalyzer cutoff=0.05
EQ	0.583	0.776	0.461	0.783
JDT	0.795	0.781	0.543	0.767
LC	0.575	0.636	0.584	0.655
ML	**0.734**	0.651	0.557	0.692*
PDE	0.684	0.682	0.566	0.717
Apache	0.714	0.689	0.635	0.717*
Safe	0.706	0.749	0.616	**0.818***
Zxing	0.605	0.619	0.530	**0.650***
ant-1.3	0.609	0.590	0.500	0.835
arc	0.670	0.611	0.523	0.701
camel-1.0	0.550	0.590	0.500	0.639
poi-1.5	0.707	0.676	0.606	0.701
redaktor	0.744	0.500	0.500	0.537
skarbonka	0.569	0.736	0.528	**0.694***
tomcat	0.778	0.746	0.640	0.818
velocity-1.4	0.725	0.609	0.500	0.391
xalan-2.4	0.755	0.658	0.499	0.751
xerces-1.2	0.624	0.453	0.500	0.489
cm1	0.653	0.622	0.551	**0.717***
mw1	0.612	0.584	0.614	0.727
pc1	0.787	0.675	0.564	0.752*
pc3	**0.794**	0.665	0.500	0.738*
pc4	**0.900**	0.773	0.589	0.682*
ar1	0.582	0.464	0.500	**0.734***
ar3	0.574	0.862	0.682	**0.823***
ar4	0.657	0.588	0.575	**0.816***
ar5	0.804	0.875	0.585	0.911*
ar6	0.654	0.611	0.527	0.640
All	0.657	0.636	0.555	**0.724***
Cliff's δ	0.143 (Negligible)	0.296 (Small)	0.792 (Large)	–

with statistical significance are underlined in the table. Between Baseline3 and our approach, better AUC values with statistical significance are shown with an asterisk (*).

The last row in Table 2 shows Cliff's δ 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 against the baseline. As explained in Section 5.5, based on a Cliff's δ, we can estimate the magnitude of the effect size. For example, the Cliff's δ between WPDP and HDP is 0.143 and its magnitude is *negligible* as in Table 2. In other words, HDP is comparable to WPDP in terms of the effect size.

We observed the following results about RQ1:

- In 25 out of 28 targets, HDP by KSAnalyzer with the cutoff of 0.05 leads to better or comparable results against WPDP with statistical significance. (The WPDP results in only ML, pc3, and pc4 are in bold font.)
- HDP by KSAnalyzer with the cutoff of 0.05 outperforms WPDP with statistical significance when considering results from all targets (*All* in the second last row in the table) together in our experimental settings.
- The Cliff's δ between WPDP and HDP is 0.143 but the magnitude of the effect size is *negligible*.
- 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)

TABLE 3: Median AUCs of baselines and HDP in KSAnalyzer (cutoff=0.05) by each source group.

Source	WPDP (Baseline1)	CPDP-CM (Baseline2)	CPDP-IFS (Baseline3)	HDP KS,0.05	Target Coverage of HDP
AEEEM	0.654	0.736	0.528	**0.739***	48%
ReLink	0.654	0.665	0.500	0.702*	88%
MORPH	0.657	0.667	0.590	**0.736***	100%
NASA	0.654	0.527	0.500	**0.734***	52%
SOFTLAB	0.695	0.612	0.554	0.708*	100%

- 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 between CPDP-CM and HDP is 0.296 and the effect size is *small*.
- 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 between CPDP-IFS and HDP is 0.792 and the magnitude of the effect size is *large*.

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 analysed 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 23 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, and CPDP-IFS. We conducted the Wilcoxon signed-rank test on results between WPDP and baselines [77]. Like Table 2, better results between baselines and our approach with statistical significance are in bold font, underlined, and/or with asterisks.

First of all, in each source group, we could observe HDP outperforms or is comparable to WPDP with statistical significance. For example, target projects were predicted by some projects in ReLink and the median AUC for HDP by KSAnalyzer is 0.702 while that of WPDP is 0.654. In addition, HDP by KSAnalyzer also outperforms or had

TABLE 4: Win/Tie/Loss results of HDP by KSAnalyzer (cutoff=0.05) against WPDP (Baseline1), CPDP-CM (Baseline2), and CPDP-IFS (Baseline3).

Target	Against								
	WPDP (Baseline1)			CPDP-CM (Baseline2)			CPDP-IFS (Baseline3)		
	Win	Tie	Loss	Win	Tie	Loss	Win	Tie	Loss
EQ	4	0	0	2	2	0	4	0	0
JDT	0	0	5	3	0	2	5	0	0
LC	6	0	1	3	3	1	3	1	3
ML	0	0	6	4	2	0	6	0	0
PDE	3	0	2	2	0	3	5	0	0
Apache	6	0	5	8	1	2	9	0	2
Safe	14	0	3	12	0	5	15	0	2
Zxing	8	0	0	6	0	2	7	0	1
ant-1.3	6	0	1	6	0	1	5	0	2
arc	3	1	0	3	0	1	4	0	0
camel-1.0	3	0	2	3	0	2	4	0	1
poi-1.5	2	0	2	3	0	1	2	0	2
redaktor	0	0	4	2	0	2	3	0	1
skarbonka	11	0	0	4	0	7	9	0	2
tomcat	2	0	0	1	1	0	2	0	0
velocity-1.4	0	0	3	0	0	3	0	0	3
xalan-2.4	0	0	1	1	0	0	1	0	0
xerces-1.2	0	0	3	3	0	0	1	0	2
cm1	7	1	2	8	0	2	9	0	1
mw1	5	0	1	4	0	2	4	0	2
pc1	1	0	5	5	0	1	6	0	0
pc3	0	0	7	7	0	0	7	0	0
pc4	0	0	7	2	0	5	7	0	0
ar1	14	0	1	14	0	1	11	0	4
ar3	15	0	0	5	0	10	10	2	3
ar4	16	0	0	14	1	1	15	0	1
ar5	14	0	4	14	0	4	16	0	2
ar6	7	1	7	8	4	3	12	0	3
Total	147 66.2%	3 1.4%	72 32.4%	147 66.2%	14 6.3%	61 27.5%	182 82.0%	3 1.3%	37 16.7%

a comparable prediction performance against CPDP-CM. 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 all source groups.

The target prediction coverage in the MORPH and SOFTLAB groups yielded 100% 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 NASA groups have 48%, 88%, and 52% 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 [79]: (1) checking prediction feasibility between source and target datasets, and (2) predicting defects.

6.3 Win/Tie/Loss Results

To investigate our performance evaluation 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), and CPDP-IFS (Baseline3) in Table 4.

KSAnalyzer with the cutoff of 0.05 conducted 222 out of 600 prediction combinations since 378 combinations do not have any matched metrics because of the cutoff threshold. In Table 4, the target dataset, EQ, was predicted in four prediction combinations and our approach, HDP, outperforms Baseline1 and Baseline3 in the four combinations (i.e. 4 Wins). However, HDP outperforms Baseline2 in only two combinations of the target, EQ (2 Wins).

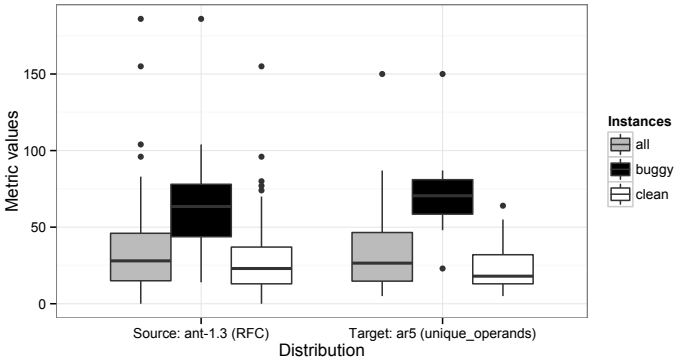

Fig. 4: Distribution of metrics (matching score=0.91) from ant-1.3⇒ar5 (AUC=0.946).

Against Baseline1, the six targets such as EQ, Zxing, skarbonka, tomcat, ar3, and ar4 have only Win results. In other words, defects in those six targets could be predicted better by other source projects using HDP models by KSAnalyzer compared to WPDP models.

In Figure 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 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.

Figure 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 Figure 4. In both datasets, approximately 75% or more buggy and clean instances will be predicted correctly. In Figure 4, the matched metrics in ant-1.3⇒ar5 are the response for class (*RFC*: number of methods invoked by a class) [80] 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 Figure 4. Typical defect prediction metrics have tendencies in which higher complexity causes more bug-proneness [1], [2], [6]. In Figure 4, instances with higher values of *RFC* and *unique_operands* have the tendency to be more bug-prone. For this reason, the model using the matched metrics could achieve such a high AUC (0.938). We could observe this bug-proneness tendency in other Win results. Since matching metrics is based on similarity of source and target metric distributions, HDP also addresses several issues related to a dataset shift such as the covariate shift and domain shift

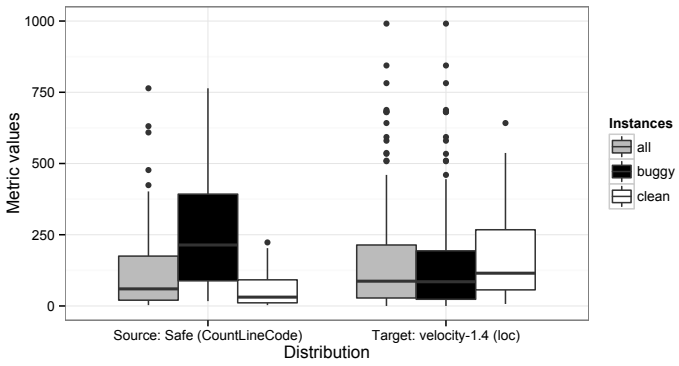

Fig. 5: Distribution of metrics (matching score=0.45) from Safe⇒velocity-1.4 (AUC=0.391).

discussed by Turhan [81].

The Win/Tie/Loss results show that with our HDP model by KSAnalyzer there is a higher possibility of getting a better prediction performance.

However, there are still about 32% Loss results against WPDP as shown in Table 4. The eight targets such as JDT, ML, redaktor, velocity-1.4, xalan-2.4, xerces-1.2, pc3, and pc4 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 bug-proneness between source and target metrics. Figure 5 shows how the bug-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 bug-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 bug-proneness in velocity-1.4. This inconsistent tendency of bug-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 bug-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.

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

TABLE 5: Prediction performance (a median AUC and % of Win) in different metric selections.

Approach	Against						HDP
	WPDP		CPDP-CM		CPDP-IFS		
	AUC	Win%	AUC	Win%	AUC	Win%	AUC
Gain Ratio	0.657	63.7%	0.645	63.2%	0.536	80.2%	0.720
Chi-Square	0.657	64.7%	0.651	66.4%	0.556	82.3%	0.727
Significance	0.657	66.2%	0.636	66.2%	0.553	82.0%	0.724
Relief-F	0.670	57.0%	0.657	63.1%	0.543	80.5%	0.709
None	0.657	47.3%	0.624	50.3%	0.536	66.3%	0.663

TABLE 6: Prediction performance in other analyzers with the matching score cutoffs, 0.05 and 0.90. (TgtCov=Target coverage)

Analyzer	Cutoff	Against						HDP	Tgt Cov
		WPDP		CPDP-CM		CPDP-IFS			
		AUC	Win%	AUC	Win%	AUC	Win%	AUC	
P	0.05	**0.684**	30.3%	0.640	45.2%	0.511	54.5%	0.617*	100%
P	0.90	0.657	54.2%	0.622	65.1%	0.535	78.3%	0.692*	96%
KS	0.05	0.657	66.2%	0.636	66.2%	0.553	82.4%	**0.724***	100%
KS	0.90	0.657	100%	0.761	71.4%	0.624	100.0%	**0.852***	21%
SCo	0.05	**0.684**	28.5%	0.640	37.3%	0.511	46.3%	0.542*	100%
SCo	0.90	**0.684**	29.0%	0.639	36.6%	0.511	48.4%	0.547	100%

Overall, the numbers of Win and Tie results are 147 and 3 respectively out of all of the 222 prediction combinations. This means that in 67.6% of prediction combinations our HDP models achieve better or comparable prediction performance than those in WPDP.

The Win/Tie/Loss results against Baseline2 and Baseline3 show a similar trend. In the 161 (72.5%) out of 222 prediction combinations, HDP outperforms and is comparable to CPDP-CM. Against Baseline3, 185 (83.3%) prediction combinations are Win or Tie results.

6.4 Performance in Different Metric Selections

Table 5 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, or CPDP-IFS, 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% in most cases against WPDP and CPDP-CM. Against CPDP-IFS, the Win results of HDP account for more than 80% after appying the metric selection approaches. This implies that the metric selection approaches can remove irrelevant metrics to build a better prediction model. In addition, this result confirms the previous studies that we can build prediction models better than or comparable to WPDP models with even a few key metrics [44], [50]. However, the percentages of Win results in 'None' were lower than those in applying metric selection. Among metric selection approaches, 'Chi-Square' and 'Significance' based approaches lead to the best performance in terms of the percentages of the Win results (64.7%-66.2%) against WPDP.

6.5 Performance in Various Metric Matching Analyzers

In Table 6, we compare the prediction performance in other analyzers with the matching score cutoff thresholds, 0.05

TABLE 7: Win/Tie percentages of HDP by KSAnalyzer (cutoff=0.05) against WPDP, CPDP-CM, and CPDP-IFS by different machine learners.

HDP Learners	Against					
	WPDP		CPDP-CM		CPDP-IFS	
	Win	Tie	Win	Tie	Win	Tie
Logistic	66.2%	1.4%	66.2%	6.3%	82.0%	2.7%
RandomForest	10.4%	2.3%	42.3%	1.4%	65.8%	2.2%
BayesNet	34.7%	4.1%	45.9%	2.7%	66.2%	2.7%
SVM	24.3%	23.0%	27.5%	0.0%	32.9%	14.9%
J48	30.2%	11.7%	32.4%	1.4%	41.9%	12.1%
SimpleLogistic	45.5%	2.7%	69.4%	6.8%	84.2%	3.2%
LMT	42.8%	3.2%	64.4%	6.3%	79.7%	3.2%

and 0.90. HDP's prediction results by PAnalyzer, with a cutoff of 0.90, are comparable to WPDP. This implies that comparing 9 percentiles between source and target metrics can evaluate the similarity of them well with a threshold of 0.90. However, PAnalyzer with the cutoff of 0.90 did not achieve 100% target coverage and is too simple an approach to lead to better prediction performance than KSAnalyzer. In KSAnalyzer with a cutoff of 0.05, the AUC (0.724) outperforms it (0.657) in WPDP with statistical significance.

HDP by KSAnalyzer with a cutoff of 0.90 could lead to significant improvement in the AUC value (0.852) compared to that (0.724) with the cutoff of 0.05. However, the target coverage is just 21%. 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% 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 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 improved from 0.617 to 0.692 and 0.724 to 0.852 in AUC, respectively. This means these two analyzers can filter out negative prediction combinations well. As a result, the percentages of Win results are also significantly increased.

Results by SCoAnalyzer were worse than WPDP results. 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.

6.6 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 Random Forest, BayesNet, SVM, J48 Decision Tree, Simple Logistic, and Logistic Model Trees (LMT) [1], [7], [8], [70], [71], [82]. Table 7 shows Win/Tie results.

Logistic Regression (Logistic) led to the best results among various learners. The Logistic Regression models works well when there is a linear relationship between a predictor variable (a metric) and the logit transformation of the outcome variable (bug-proneness) [83]. In our study, this linear relationship is related to the bug-proneness tendency of a metric, that is, a higher complexity causes more bug-proneness [1], [2], [6]. As the consistent bug-prone tendency of matched metrics is important in HDP, the HDP models built by Logistic Regression can lead to the best prediction performance.

HDP models built by other learners such as Simple Logistic and LMT led to comparable results to Logistic Regression against CPDP-CM and CPDP-IFS. Against Baseline2, Win results (69.4% and 64.4%) in Simple Logistic and LMT are comparable to Win results (66.2%) in Logistic. Simple Logistic also uses the logit function and LMT adopts Logistic Regression at the leaves of decision tree [82]. In other words, both learners consider the linear relationship like Logistic Regression [83]. In our experimental settings, HDP tends to work well with the learners based on the linear relationship between a metric and a label (bug-proneness).

6.7 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 WPDP, CPDP-CM, and CPDP-IFS when using KSAnalyzer with the cutoff of 0.05 and significance attribute selection.
- Compared to WPDP, HDP achieved 67.6% of Win/Tie results. However, there are still 32.4% 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 compare distributions of two variables [55], [58].
- HDP worked well with Logistic Regression but not other machine learners. One possible reason is that Logistic Regression captures the linear relationship between metrics and the logit transformation of labels that is related to the bug-proneness tendency of the metrics.

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?

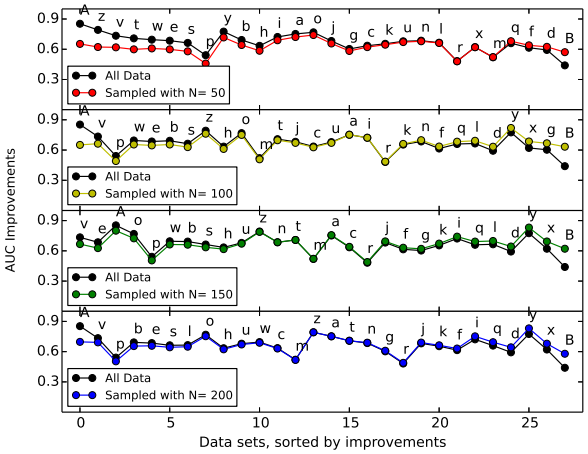

Fig. 6: Improvements of using sampled data over all data with sampled size N = {50, 100, 150, 200}. We label the data in table 1 from a to z, and the last two datasets ar5 and ar6 as A and B.

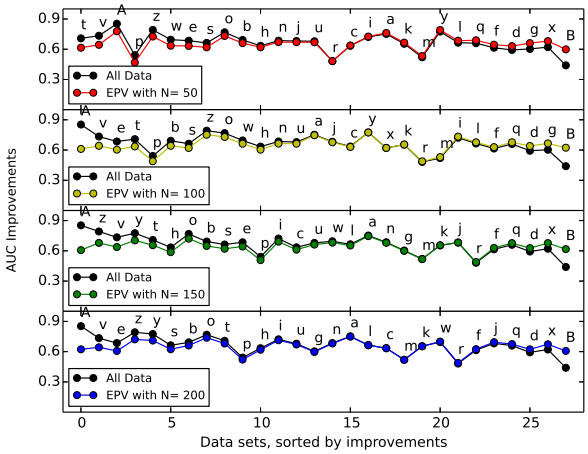

Fig. 7: Improvements of using sampled data N = {50, 100, 150, 200} with EPV constraints. We label the data in table 1 from a to z, and the last two datasets ar5 and ar6 as A and B.

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], [12], [13], [39], [45], the basic HDP method we proposed above uses *all* the instances in potential source and target data to perform KS-test 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* data from source 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 [46] and CCA+ [47]. 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 28 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.

7.1 200 Random Samples are Enough

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 to generate smaller datasets of size $N \in \{50, 100, 150, 200\}$. If the number of instances in the original dataset is smaller than N, all those instances will be included. With those sampled data, we perform metric matching to build a learner and finally predict labels of all original data in the target project.

The results for this HDP-with-limited-data experiment is shown in Figure 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 $N = 200$, $27/28$ tests are within 0.05 difference in terms of AUC and $17/28$ tests show smaller datasets have equivalent or even better performance. Here, we recommend that sample size $N = 200$ could be good enough for this HDP framework to obtain a good predictor.

7.2 20 Defective Examples are Enough

The results of the last section are very encouraging—a small number of source and target examples are enough

for effective transfer learning. Naturally, we were suspicious of this result (since it was "almost too good to be true"). Accordingly, we explored the literature and found:

- Evidence that this "a few examples are enough" has been seen in other domains [84];
- Methods to reduce the cost of sampling this dataset, even further.

Working in the domain cardiology, Peduzzi et al. [84] report 10 "events" per variable (EPV) are enough to maximize the predictive performance of Logistic Regression (the learner used in this study). Translated into our terminology, that study would predict that 10 defective data per independent variable should suffice for effective learning. Of course, learners require defective and *non-defective* examples so to M defective examples, we add $(N - M)$ non-defective examples more.

To apply the method of Peduzzi et al., we first instrumented HDP to determine how many independent variables were picked during *metric selection*. In practice, that number was very small: usually just 2. Hence, applying Peduzzi et al.'s rule (EPV=10), we picked $2 \times 10 = 20$ randomly selected defective data, then randomly added $(N - M)$ non-defective data for $N \in \{50, 100, 150, 250\}$ for the source data. For the target dataset, since we do not know the data labels, we simply sampled N for target data in metric matching.

(Aside: note that this is different to the above experiment since, before, the more N examples we selected, the more *defective* instances we would use. Now, in this experiment, we will also use a fixed $M = 20$ number of defects.).

The results are shown in Figure 7. Like before:

- These are median AUC results from 20 repeats;
- The *black/colourful* lines show the results using *all/some* of data from the source and target datasets during metric matching and learner building, respectively (but now, the *some* never contains more than $M = 20$ defective instances);
- The datasets are ordered left to right according the performance difference between using *all* or *some* of the data;
- On the left/right , we do *better/worse* using *all* data than using *some*.

We observe that between $N = 50$ and $N = 200$, the performance delta does not change by much. Also, if we compare $N = 200$ between Figure 6 and Figure 7, there is no large disadvantage of using just $M = 20$ defective instances.

Note that such a very small sample would be quick to collect: after writing (say) a few hundred classes, inspect enough to find 20 defective ones—at which point that data is a candidate for transfer learning.

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:

- 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 about defect datasets

Recent results [45], [85] 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 [85].

Other results [86] 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 [53] 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 [87] to divide dimensions accroding to some *descritization policy* [88]. 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 [89], 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 1000 times of iterations, we picked k, d, b, e values at random from:

```
dimensions = (1,2)
|   dimensions = 1
|   |   e ≤ 0.1
|   |   |   bins = (1,2,3) : n < 50
|   |   |   bins > 3       : n < 100
|   |   e > 0.1 : n < 50
|   dimensions > 1
|   |   bins = (1,2,3)
|   |   |   e = 0.1 : n < 100
|   |   |   e > 0.1 : n < 50
|   |   bins > 3
|   |   |   bins = (4,5)
|   |   |   |   e = 0.1 : n < 200
|   |   |   |   e > 0.1
|   |   |   |   |   e ≤ 0.2
|   |   |   |   |   |   bins = 4 : n < 100
|   |   |   |   |   |   bins = 5 : n < 200
|   |   |   |   |   e > 0.2      : n < 100
|   |   |   |   bins > 5
|   |   |   |   |   e ≤ 0.2 : n ≥ 200
|   |   |   |   |   e > 0.2 : n < 200
dimensions > 2
|   bins = (1,2)
|   |   dimensions = (3,4,5)
|   |   |   dimensions = 3
|   |   |   |   e ≤ 0.2 : n < 200
|   |   |   |   e > 0.2 : n < 50
|   |   |   dimensions = (4,5)
|   |   |   |   e = 0.1 : n < 200
|   |   |   |   e > 0.1
|   |   |   |   |   dimensions = 4 : n < 100
|   |   |   |   |   dimensions = 5
|   |   |   |   |   |   e ≤ 0.3 : n < 200
|   |   |   |   |   |   e > 0.3 : n < 100
|   |   dimensions > 5 : n ≥ 200
|   bins > 2
|   |   dimensions = 3
|   |   |   bins = (3,4)
|   |   |   |   e ≤ 0.3
|   |   |   |   |   bins = 3
|   |   |   |   |   |   e = 0.1 : n ≥ 200
|   |   |   |   |   |   e > 0.1 : n < 200
|   |   |   |   |   bins = 4 : n ≥ 200
|   |   |   |   e > 0.3 : n < 200
|   |   |   bins > 4 : n ≥ 200
|   |   dimensions > 3 : n ≥ 200
```

Fig. 8: How many n examples are required to be at least 67% likely to find defects occurring at probability e.

- $k \in \{1, 2, 3, 4, 5\}$;
- $d \in \{3, 4, 5, 6, 7\}$ 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 $d \leq 7$ of those dimensions [2].)

Within 1000 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.

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% 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%;
- F-measures (i.e. the harmonic mean of recall and precision) of 95%

8.2 Results

The resulting decision tree, shown in Figure 8, 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% 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, 50 to 100 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 Figure 8 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%) but far fewer when the signal occurs at $e > 10\%$.
- 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:

- Other researchers working with Logistic Regression [84] have reported that 10 "events per decision variable" are adequate for building models using that data miner. For our defect datasets, in Figure 7, we repeated the analysis of [84] and found that considering 2 independent variable on average, there was little improvement after 100 examples with EPV = 10 (as would have been predicted by [84]).
- The maths of Section 8 show that this "100 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], [12], [37], [38], [39], [40]. 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 6, we confirm that HDP by KSAnalyzer with the cutoff of 0.05 outperforms WPDP and shows 100% 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 [78], 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 6 shows KSAnalyzer with the cutoff of 0.90 led to 100% Win results in feasible predictions against WPDP.

9.2 Threats to Validity

We evaluated our HDP models in AUC. AUC is known as a good measure for comparing different prediction models [13], [69], [70], [71]. 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 50:50 random splits 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 50:50 random sampling. 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.

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 (HDP) based on metric matching using statistical analysis [55]. 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 study, we suggested that the sample size of 200 for source (with at least 20 defective samples) and target datasets could be effective enough for our HDP models.

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, we will explore the feasibility of building various prediction and recommendation models using heterogeneous datasets.

REFERENCES

[1] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, 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, pp. 2–13, January 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, pp. 751–761, October 1996.

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

[5] T. McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, vol. SE-2, no. 4, pp. 308–320, Dec 1976.

[6] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proceedings of the 2013 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 432–441.

[7] T. Lee, J. Nam, D. Han, S. Kim, and I. P. Hoh, "Micro interaction metrics for defect prediction," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2011.

[8] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings of the 2013 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013, pp. 382–391.

[9] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. New York, NY, USA: ACM, 2009, pp. 91–100.

[10] 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 *Proceedings of the 11th Working Conference on Mining Software Repositories*. New York, NY, USA: ACM, 2014, pp. 172–181.

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

[12] 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.

[13] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the "imprecision" of cross-project defect prediction," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. New York, NY, USA: ACM, 2012.

[14] 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, October 2009.

[15] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. (2012, June) The promise repository of empirical software engineering data. [Online]. Available: http://promisedata.googlecode.com

[16] J. Nam and S. Kim, "Heterogeneous defect prediction," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 508–519. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786814

[17] IEEE-1012, "IEEE standard 1012-2004 for software verification and validation," 1998.

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

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

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

[21] D. Budgen, 2006, keynote address, CSEET'06.

[22] B. K. David Budgen, Pearl Brereton, "Is evidence based software engineering mature enough for practice & policy?" in *33rd Annual IEEE Software Engineering Workshop 2009 (SEW-33), Skovde, Sweden*, 2009.

[23] D. Posnett, V. Filkov, and P. Devanbu, "Ecological inference in empirical software engineering," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 362–371. [Online]. Available: http://dx.doi.org/10.1109/ASE.2011.6100074

[24] Y. Yang, L. Xie, Z. He, Q. Li, V. Nguyen, B. W. Boehm, and R. Valerdi, "Local bias and its impacts on the performance of parametric estimation models," in *PROMISE*, 2011.

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

[26] T. Menzies, A. Butcher, D. R. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann, "Local versus global lessons for defect prediction and effort estimation," *IEEE Trans. Software Eng.*, vol. 39, no. 6, pp. 822–834, 2013, available from http://menzies.us/pdf/12localb.pdf.

[27] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *ICSE '05*. ACM, 2005, pp. 580–586.

[28] T. Menzies, D. Raffo, S. Setamanit, Y. Hu, and S. Tootoonian, "Model-based tests of truisms," in *ASE '02*, 2002, available from http://menzies.us/pdf/02truisms.pdf.

[29] 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 *ICSE '13*. IEEE, 2013, pp. 372–381.

[30] S. Rakitin, *Software Verification and Validation for Practitioners and Managers, Second Edition*. Artech House, 2001.

[31] 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, available from http://menzies.us/pdf/06learnPredict.pdf.

[32] A. Tosun, A. Bener, and R. Kale, "AI-based software defect predictors: Applications and benefits in a case study," in *IAAI*, 2010.

[33] A. Tosun, A. Bener, and B. Turhan, "Practical considerations of deploying ai in defect prediction: A case study within the Turkish telecommunication industry," in *PROMISE'09*, 2009.

[34] F. Shull, V. B. ad B. Boehm, A. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz, "What we have learned about fighting defects," in *Proceedings of 8th International Software Metrics Symposium, Ottawa, Canada*. IEEE, 2002, pp. 249–258.

[35] M. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, 1976.

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

[37] G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Multi-objective cross-project defect prediction," in *Software Testing, Verification and Validation, 2013 IEEE Sixth International Conference on*, March 2013.

[38] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'union fait la force," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, Feb 2014, pp. 164–173.

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

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

[41] Y. Zhang, D. Lo, X. Xia, and J. Sun, "An empirical study of classifier combination for cross-project defect prediction," in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 2, July 2015, pp. 264–269.

[42] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter languagereuse," in *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*. New York, NY, USA: ACM, 2008, pp. 19–24.

[43] Y. Kamei, E. Shihab, B. Adams, A. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *Software Engineering, IEEE Transactions on*, vol. 39, no. 6, pp. 757–773, June 2013.

[44] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, no. 0, pp. 170 – 190, 2015.

[45] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 182–191. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597078

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

[47] X. Jing, F. Wu, X. Dong, F. Qi, and B. Xu, "Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 496–507. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786813

[48] E. Shihab, "Practical software quality prediction," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, Sept 2014, pp. 639–644.

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

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

[51] S. Shivaji, E. J. Whitehead, R. Akella, and S. Kim, "Reducing features to improve code change-based bug prediction," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 552–569, 2013.

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

[53] M. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 15, no. 6, pp. 1437–1447, Nov 2003.

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

[55] F. J. Massey, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American Statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

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

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

[58] H. W. Lilliefors, "On the kolmogorov-smirnov test for normality with mean and variance unknown," *Journal of the American Statistical Association*, vol. 62, no. 318, pp. pp. 399–402, 1967.

[59] R. Wu, H. Zhang, S. Kim, and S. Cheung, "Relink: Recovering links between bugs and changes," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2011.

[60] F. Peters and T. Menzies, "Privacy and utility for defect prediction: experiments with morph," in *Proceedings of the 2012 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2012, pp. 189–199.

[61] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *Software Engineering, IEEE Transactions on*, vol. 39, no. 9, pp. 1208–1215, Sept 2013.

[62] Understand 2.0, "http://www.scitools.com/products/."

[63] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, pp. 13–23.

[64] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, "High-impact defects: a study of breakage and surprise defects," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. New York, NY, USA: ACM, 2011, pp. 300–310.

[65] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, pp. 10–18, November 2009.

[66] 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 *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*. New York, NY, USA: ACM, 2010, pp. 119–128.

[67] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. New York, NY, USA: ACM, 2008, pp. 2–12.

[68] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering*. New York, NY, USA: ACM, 2011, pp. 1–10.

[69] E. Giger, M. D'Ambros, M. Pinzger, and H. C. Gall, "Method-level bug prediction," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. New York, NY, USA: ACM, 2012, pp. 171–180.

[70] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *Software Engineering, IEEE Transactions on*, vol. 34, no. 4, pp. 485–496, 2008.

[71] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 356–370, 2011.

[72] T. Mende, "Replication of defect prediction studies: Problems, pitfalls and recommendations," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. New York, NY, USA: ACM, 2010, pp. 5:1–5:10.

[73] 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 *annual meeting of the Florida Association of Institutional Research, February*, 2006, pp. 1–3.

[74] 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," *Software Engineering, IEEE Transactions on*, vol. 39, no. 8, pp. 1040–1053, 2013.

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

[76] G. Valentini and T. G. Dietterich, "Low bias bagged support vector machines," in *Proceedings of the Twentieth International Conference on Machine Learning*. AAAI Press, 2003, pp. 752–759.

[77] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, Dec. 1945.

[78] G. W. Corder and D. I. Foreman, *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. New Jersey: Wiley, 2009.

[79] D. Kim, Y. Tao, S. Kim, and A. Zeller, "Where should we fix this bug? a two-phase recommendation model," *Software Engineering, IEEE Transactions on*, vol. 39, no. 11, pp. 1597–1610, Nov 2013.

[80] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, pp. 476–493, June 1994.

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

[82] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proc. of the 37th Int'l Conf. on Software Engineering (ICSE)*, ser. ICSE '15, 2015, pp. 789–800.

[83] J. Bruin. (2011, Feb.) newtest: command to compute new test, http://www.ats.ucla.edu/stat/stata/ado/analysis/.

[84] P. Peduzzi, J. Concato, E. Kemper, T. R. Holford, and A. R. Feinstein, "A simulation study of the number of events per variable in logistic regression analysis," *Journal of clinical epidemiology*, vol. 49, no. 12, pp. 1373–1379, 1996.

[85] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets," in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015)*, 2015.
[86] M. Shepperd and D. C. Ince, "A Critique of Three Metrics," *The Journal of Systems and Software*, vol. 26, no. 3, pp. 197–210, sep 1994.
[87] T. Menzies, "Data Mining," in *Recommendation Systems in Software Engineering*, 2014, pp. 39–75.
[88] J. Lustgarten, V. Gopalakrishnan, H. Grover, and S. Visweswaran, "Improving classification performance with discretization on biomedical datasets," pp. 445–449, 2008.
[89] J. M. Voas and K. W. Miller, "Software testability: The new verification," *IEEE software*, no. 3, pp. 17–28, 1995.