Benchmarking cross-project defect prediction approaches with costs metrics

Steffen Herbold
University of Goettingen, Institute of Computer Science
Göttingen, Germany
herbold@cs.uni-goettingen.de

ABSTRACT
Defect prediction can be a powerful tool to guide the use of quality assurance resources. In recent years, many researchers focused on the problem of Cross-Project Defect Prediction (CPDP), i.e., the creation of prediction models based on training data from other projects. However, only few of the published papers evaluate the cost efficiency of predictions, i.e., if they save costs if they are used to guide quality assurance efforts. Within this paper, we provide a benchmark of 26 CPDP approaches based on cost metrics. Our benchmark shows that trivially assuming everything as defective is on average better than CPDP under cost considerations. Moreover, we show that our ranking of approaches using cost metrics is uncorrelated to a ranking based on metrics that do not directly consider costs. These findings show that we must put more effort into evaluating the actual benefits of CPDP, as the current state of the art of CPDP can actually be beaten by a trivial approach in cost-oriented evaluations.

CCS CONCEPTS
• Software and its engineering → Software defect analysis; • General and reference → Experimentation;

KEYWORDS
defect prediction, cross-project, cost metrics

1 INTRODUCTION
Software defect prediction has been under investigation in our community for many years. The reason for this is the huge cost saving potential of accurately predicting in which parts of software defects are located. This information can be used to guide quality assurance efforts and ideally have fewer post release defects with less effort. Since one major problem of defect prediction approaches is the availability of training data, researchers turned towards Cross-Project Defect Prediction (CPDP), i.e., the prediction of defects based on training data from other projects. A recent benchmark by Herbold et al. [24] compared the performance of 24 suggested approaches by researchers between 2008 and 2015. However, the benchmark has one major limitation: it does not consider the impact on costs directly. Instead, Herbold et al. used the machine learning metrics AUC, F-measure, G-measure, and MCC. From a cost perspective, these metrics only make sense, if all software entities that are predicted as defective get additional quality assurance attention. Even then, the cost of missing defects would be assumed as equal to the cost of additional review effort. However, according to the literature, post-release defects cost 15-50 times more than additional quality assurance effort [36]. Rahman et al. [55] already found, that the results regarding CPDP may be very different, if considered from a cost-oriented perspective. Whether this finding by Rahman et al. translates to the majority of the CPDP literature is unclear, because a cost-sensitive evaluation is missing.

We close this gap with this paper. Our contribution is an adoption of the benchmark from Herbold et al. [24] for cost metrics to determine a cost-sensitive ranking of CPDP approaches. We compare 26 CPDP approaches published between 2008 and 2016, as well as three baselines. Our results show that a trivial baseline approach that considers all code as defective is almost never significantly outperformed by CPDP. Only one CPDP proposed by Liu et al. [40] performs better for one out of three cost metrics, but even this is only the case on one of the two data sets we use. For the other data set, the performance of not statistically significantly better. For the other two cost metrics, no CPDP approach performs statistically significantly better than the trivial baseline. Moreover, there is no correlation between ranking CPDP approaches based on cost metrics, and the ranking produced by Herbold et al. [24]. Hence, cost metrics should always be considered in addition to other metrics, because otherwise we do not evaluate if proposed models to what they should: reduce costs.

The remainder of this paper is structured as follows. We discuss related work in Section 2. Then, we introduce our benchmark methodology including the research questions, data used, performance metrics, and statistical evaluation in Section 3. Afterwards, we present our results in Section 4, followed the discussion in Section 5 and the threats to validity in Section 6. Finally, we conclude the paper in Section 7.

2 RELATED WORK
We split our discussion of the related work into two parts. First, we discuss the related work on defect prediction benchmarks. Second, we discuss the related work on CPDP.

2.1 Defect prediction benchmarks
Our benchmark on cost aspects is influenced by four defect prediction benchmarks from the literature. Lessmann et al. [39] and Ghotra et al. [18] evaluated the impact of different classifiers on Within-Project Defect Prediction (WPDP). D’Ambros et al. [14]
Table 1: Related work on CPDP included in the benchmark. Acronyms are defined following the authors and reference in the first column.

| Acronym          | Short Description                                                                 |
|------------------|-----------------------------------------------------------------------------------|
| Khoshgoftaara08  | Khoshgoftaara et al. [35] proposed majority voting of multiple classifiers trained for each product in the training data. |
| Watanabe08       | Watanabe et al. [72] proposed standardization based on mean values of the target product. |
| Turhan09         | Turhan et al. [67] proposed a log-transformation and nearest neighbor relevancy filtering. |
| Zimmermann09     | Zimmermann et al. [77] proposed a decision tree to select suitable training data. |
| CamargoCruz09    | Camargo Cruz and Ochinizu [6] proposed a log-transformation and standardization based on the median of the target product. |
| Liu10            | Liu et al. [40] proposed an S-expression tree created by genetic program.         |
| Menzies11        | Menzies et al. [43, 44] proposed local models for different regions of the training data through clustering. |
| Ma12             | Ma et al. [41] proposed data weighting using the concept of gravitation.          |
| Peters12         | Peters and Menzies [51] proposed an approach for data privacy using a randomized transformation called MORPH. |
| Uchigaki12       | Uchigaki et al. [70] proposed an ensemble of univariate logistic regression models build for each attribute separately. |
| Canfora13        | Canfora et al. [7, 8] proposed a multi-objective genetic program to build a logistic regression model that optimizes costs and the number of defects detected. |
| Peters13         | Peters et al. [52] proposed relevancy filtering using conditional probabilities combined with MORPH data privatization. |
| Herbold13        | Herbold [21] proposed relevancy filtering using distributional characteristics of products. |
| ZHe13            | Z. He et al. [20] proposed attribute selection and relevancy filtering using separability between training and target products. |
| Panichella14     | Panichella et al. [50] proposed the CODEP meta classifier over the results of multiple classification models. |
| Ryu14            | Ryu et al. [57] proposed similarity based resampling and boosting.               |
| PHe15            | P. He et al. [19] proposed feature selection based on how often the metrics are used for classification models build using the training data. |
| Peters15         | Peters et al. [53] proposed LACE2 as an extension with further privacy of CLIFF and MORPH. |
| Kawata15         | Kawata et al. [33] proposed relevancy filtering using DBSCAN clustering.         |
| YZhang15         | Y. Zhang et al. [76] proposed the ensemble classifiers average voting, maximum voting, boosting and bagging. |
| Amasaki15        | Amasaki et al. [1] proposed feature selection and relevancy filtering based on minimal metric distances between training and target data. |
| Ryu15            | Ryu et al. [58] proposed relevancy filtering based on string distances and LASER classification. |
| Nam15            | Nam and Kim [46] proposed unsupervised defect prediction based on the median of attributes. |
| Tantithamthavorn16 | Tantithamthavorn et al., 2016 [63] proposed to use hyper parameter optimization for classifiers. |
| FZhang16         | F. Zhang et al. [75] proposed unsupervised defect prediction based on spectral clustering. |
| Hosseini16       | Hosseini et al. [27, 28] proposed a genetic program and nearest neighbor relevancy filtering to select training data. |

compared different kinds of metrics and classification models for WPDP. Herbold et al. [24] compared CPDP approaches on multiple data sets using multiple performance metrics.

The benchmarks by Lessmann et al., D’Ambros et al., and Herbold et al. followed Demšar’s guidelines [15] and make use of the Friedman test [16] with the post-hoc Nemenyi test [48]. Herbold et al. extended this concept using rankscores, that allow the combination of results from multiple data sets and performance metrics into a single ranking. Ghotra et al. use a different statistical procedure based on ANOVA [16] and the Scott-Knott test [60].

Our benchmark design is similar to the design used by Herbold et al. [24]. However, there are two major differences: 1) we focus on different research questions, i.e., the performance using cost-metrics, whereas Herbold et al. focused on machine learning metrics that do not take costs into account; and 2) we extended the statistical evaluation with a recently proposed effect size correction taking pattern from ScottKnottESD proposed by Tantithamthavorn et al. [64].

2.2 Cross-project defect prediction

The scope of our benchmark is approaches that predict software defects in a target product using metric data collected from other projects. However, our benchmark does not cover the complete body of CPDP research. Specifically, we do not address the following:

- Mixed-project defect prediction, i.e., approaches that require labelled training data from the target product, e.g., [10, 59, 68, 69, 73].
- Heterogenous defect prediction, i.e., approaches that work with different metric sets for the training and target products, e.g., [29, 45].
We address the following six research questions within this study.

- Just-in-time defect prediction, i.e., defect prediction for specific commits, e.g., [17, 32].
- Approaches that require project context factors, e.g., [74].

Additionally, we excluded two works that use transfer component analysis [49] by Nam et al. [47] and by Jing et al. [30]. Transfer component analysis has major scalability issues due to a very large eigenvalue problem that needs to be solved. Herbold et al. [22] already determined that they could only compute results in less than one day for the smaller data sets for the approach by Nam et al. [47]. We tried to resolve this problem by using a scientific compute cluster, where we had access to nodes with up to 64 cores and 256 GB of memory. We were still not able to compute results for a large data set with 17681 instances before hitting the execution time limit for computational jobs of 48 hours.

This leaves us with 26 approaches that were published through 29 publications listed in Table 1. For each of these approaches, we define an acronym, which we will use hereafter to refer to the approach and a short description of the approach. The list in Table 1 is mostly consistent with the benchmark from Herbold et al. [24] on CPDP. Additionally, we extended the work by Herbold et al. with three further replications of approaches that were published in 2016, i.e., Tantithamthavorn16, FZhang16, and Hosseini16.

3 BENCHMARK METHODOLOGY

We now describe the methodology of our benchmark, including our research questions, the data we used, the baselines and classifiers, the performance metrics, and the statistical analysis.

3.1 Research Questions

We address the following six research questions within this study.

RQ1: Does it matter if we use defect counts or binary labels for cost metrics?

RQ2: Which approach performs best if quality assurance is applied according to the prediction?

RQ3: Which approach performs best if additional quality assurance can only be applied to a small portion of the code?

RQ4: Which approach performs best independent of the prediction threshold?

RQ5: Which approach performs best overall in terms of costs?

RQ6: Is the overall ranking based on cost metrics different from the overall ranking based on the \( AUC \), \( F\)-measure, \( G\)-measure, and MCC?

With RQ1, we address the question if binary labels as defective/non-defective are sufficient, or if defect counts are required to compare costs. Binary labels carry less information and should lead to less accurate results. For example, you save more costs if you prevent two defects, instead of one. The question is, does this really matter, i.e., do the values of performance metrics change significantly? This question is especially interesting, as not all publicly available defect prediction data sets provide defect counts. In case the impact of defect counts is large, we may only use data sets that provide this information for our benchmark.

With research questions RQ2-RQ4 we consider different quality assurance scenarios. RQ2 explores the costs savings, if someone trusts the defect prediction model completely, i.e., applies quality assurance measures exactly according to the prediction of the model. RQ3 considers the case where the defect prediction model is used to identify a small portion of the code for additional quality assurance, i.e., a setting with a limited quality assurance budget. RQ2 and RQ3 have specific prediction thresholds, i.e., a certain amount of the predicted defects are considered. With RQ4, we provide a threshold independent view on costs, which is valuable for selecting an approach if the amount of effort to be invested is unclear beforehand. To provide a general purpose ranking, we throw all the considerations from RQ2-RQ4 together for RQ5 and evaluate which approach performs best if all criteria are considered. Thus, RQ5 provides an evaluation that accounts for different application and cost scenarios. With RQ6 we address the question if the usually used machine learning metrics are sufficient to estimate a cost-sensitive ranking, i.e., if they produce a similar or a different ranking from directly using cost metrics.

3.2 Data

We use data from two different defect prediction data sets from the literature for our benchmark listed in Table 2. These data sets are a subset of the data sets that Herbold et al. [24] used in their benchmark. The other three data sets that Herbold et al. used could not be used for different reasons. The MDP and RELINK data were infeasible due to our results regarding RQ1 (see Section 4). The NETGENE data does not contain the size of artifacts, which is required for cost-sensitive evaluations.

We can only give a brief summary of both data sets due to space restrictions. Full lists of the software metrics, products contained, etc. can be found in the literature at the cited references for each data set or summarized in the benchmark by Herbold et al. [24].

The first data set was donated by Jureczko and Madeyski [31] and consists of 48 product releases of 15 open source projects, 27 product releases of six proprietary projects and 17 academic products that were implemented by students. For each of these releases, 20 static product metrics for Java classes, as well as the number of defects are part of the data. Taking pattern from Herbold et al. [22], we use 62 of the products. We do not use the 27 proprietary products to avoid threats to the validity of our results due to mixing proprietary and open source software. Moreover, three of the academic products contain less than five defective instances, which is too few for reasonable analysis with machine learning. In the following, we will refer to this data set as JURECZKO.

The second data set was published by D’Ambros et al. [13] and consists of five software releases from different projects. For each of

| Name     | #Products | #Instances | #Defective |
|----------|-----------|------------|------------|
| JURECZKO | 62        | 17681      | 6062       |
| AEEEM    | 5         | 5371       | 893        |

Table 2: Summary of used data sets.

Footnotes:
1 JURECZKO as defined in Section 3.2
2 According to the paper by Herzig et al. [26], complexity and size should be included in the data. There are archives called complexity_diff.tar.gz available for each product in the data. However, the archives seem to contain the differences for complexity and size metrics for each transaction identified by a revision hash. We did not find absolute values for the size, which would be required for cost metrics.
3 The data is publicly available online: http://openscience.us/repo/defect/ck/ (last checked: 2017-08-25).
4 The data is publicly available online: http://bug.inf.uni.ch/ (last checked: 2017-08-25).
these releases, 71 software metrics for Java classes are available, that include static product metrics and process metrics, like weighted churn and linearly decayed entropy, as well as the number of defects. In the following, we refer to this data set as AEEEM, taking pattern from Nam et al. [47].

### 3.3 Baselines and Classifiers

Because we adopt Herbold et al.’s [24] benchmark methodology for using cost metrics, our choices for baselines and classifiers are nearly identical. We adopt three performance baselines from Herbold et al. [24] that define naive approaches for classification models: ALL that takes all available training data as is, RANDOM which randomly classifies instances as defective with a probability of 0.5, and FIX which classifies all instances as defective. We do not adopt the baseline CV for 10x10 cross-validation, because cross-validation is not implementable in practice and is known to overestimate the performance of WPDP [62]. This may skew rankings, as approaches may be outperformed by something that overestimates performance. Moreover, cross-validation is an estimator for WPDP performance and, therefore, out of scope of our benchmark for CPDP models.

We use a C4.5 decision tree (DT) [54], logistic regression (LR) [12], naïve bayes (NB) [56], random forest (RF) [3], RBF network (NET) [4, 9], and a support vector machine with radial basis function kernel (SVM) [71] for all approaches that did not propose a classifier, but cause it performs best in the evaluation by Tantithamthavorn et al.

Additionally, we train a C5.0 decision tree [37] with hyper parameter optimization, suggested by Tantithamthavorn et al. [63]. For DT, NET, and LR no hyper parameters to optimize are contained in the caret R package [38] suggested by Tantithamthavorn et al. Additionally, we train a C5.0 decision tree [37] with hyper parameter optimization, because it performs best in the evaluation by Tantithamthavorn et al.

We refer to these as optimized classifiers as NBCARET, RFCARET, SVMCARET and C50CARET.

The remaining eight of the approaches directly propose a classification scheme, which we use:

- genetic program (GP) for Liu10;
- logistic ensemble (LE) for Uchigakii12;
- MODEP for Canfora13;
- CODEP with Logistic Regression (CODEP-LR) and CODEP with a Bayesian Network (CODEP-BN) for Panichella14;
- the value-cognitive boosted SVM (VCBSVM) for Ryu14;
- average voting (AVG VOTE), maximum voting (MAX VOTE), bagging with a C4.5 Decision Tree (BAG-DT), bagging with Naïve Bayes (BAG-NB), boosting with a C4.5 Decision Tree (BOOST-DT), boosting with Naïve Bayes (BOOST-NB) for YZhang15;
- spectral clustering (SC) for FZhang16; and
- search-based selection (SBS) for Hosseini16.

The MODEP classifier by Canfora13 requires either a constraint with a desired recall, or a desired cost objective. Herbold et al. [24] decided to use a recall of 0.7 as the constraint. In our benchmark, we sample different values for recall and use the values 0.1 to 1.0 in steps of 0.1. We denote the different recall constraints after the classifier name using the percentage, e.g., MODEP10 for the constraint recall=0.1.

To deal with randomization, we repeat all approaches that contain random components 10 times and then use the mean value of these repetitions for comparison with the other approaches. Following Herbold et al. [24], these are Liu10, Canfora10, Menzies11, Peters12, Peters13, ZHe13, Peters15, as well as the baseline RANDOM. Moreover, two of the three approaches that we added to the benchmark contain random components, i.e., Hosseini16 because of the random test data splits and the genetic program and Tantithamthavorn16 because of the cross-validation for the hyper parameter optimization.

### 3.4 Performance Metrics

To properly evaluate our research questions, we require metrics that measure the cost for different settings. In order to not re-invent the wheel, we scanned the literature and found fitting metrics for all our research questions.

For RQ2, we need a metric that can be used to evaluate the costs if one follows the classification achieved with a defect prediction model, i.e., to apply quality assurance of everything that is predicted as defective and nothing else. We use the metric \( \text{NECM}_{\text{ratio}} \), which is defined as

\[
\text{NECM}_{\text{ratio}} = \frac{fp + C_{\text{ratio}} \cdot fn}{tp + fp + tn + fn}
\]  

This metric was, e.g., used by Liu et al. [40] and Khoshgoftaar et al. [35] and measures the costs resulting from overhead in quality assurance effort through false positive predictions versus the costs due to missed defects through false negative predictions. \( C_{\text{ratio}} \) is used to define the difference in cost for false positive and false negative predictions. Khoshgoftaar et al. and Liu et al. both use 15, 20, and 25 as values for \( C_{\text{ratio}} \). The cost of missing defects may be 15-50 times higher than that of additional quality assurance measures according to the literature [36]. For our benchmark, we use \( C_{\text{ratio}} = 15 \), i.e., the most conservative cost scenario, where reviews are relatively expensive in comparison to the saved costs of finding a defect through the prediction model.

To evaluate RQ3, i.e., the costs when only a small part of the code shall be reviewed, we use the metric \( \text{RelB}_{20\%} \), defined as the percentage of defects found when inspecting 20% of the code. Thus, the defect prediction model is used to rank all code entities. Then, the entities are considered starting with the highest ranked entity until 20% of the code is covered. This metric is an adoption of the metric \( \text{NoB}_{20\%} \) used by Y. Zhang et al. [76]. The only difference is that Y. Zhang et al. consider absolute numbers, whereas we consider the percentage. This difference is required due to the diversity in the size of software products. If we do not remove the strong impact of the project size from the values for this metric and use absolutes instead of ratios, our statistical analysis would be strongly
influenced by the size of the products and not measure the actual defect detection capability.

The metrics $N_{ECM_{15}}$ and $RelB_{20\%}$ evaluate the CPDP models in such a way that a fixed amount of code is considered for additional quality assurance. For $N_{ECM_{15}}$, we follow the classification, i.e., we consider the scenario that is most likely according to the predictions of the CPDP model. For $RelB_{20\%}$, we stop at 20\% of the code, regardless of the scores of the classification model. This is related to the notion of thresholds for classification in the machine learning world: a learned prediction model has a scoring function as output and everything above a certain threshold is then classified as defective. While there are good reasons to use the strategies the thresholds are picked by $N_{ECM_{15}}$ and $RelB_{20\%}$, these thresholds are still magic numbers. To evaluate RQ4, we use the threshold independent metric AUCEC, which is defined as the area under the curve of review effort versus number of defects found [55]. This is a threshold independent metric that analyzes the performance of a defect likelihood ranking produced by a classifier for all possible thresholds. Thus, the value of AUCEC is not a measure for a single prediction model with a fixed threshold like $N_{ECM_{15}}$ and $RelB_{20\%}$, but instead for a family of prediction models with all possible threshold values. Regarding costs, this means that AUCEC evaluates the costs for all possible amounts of code that to which additional quality assurance is applied, starting from applying no quality assurance at all and stopping with applying quality assurance to the complete product.

To evaluate RQ1 and RQS, we use the metrics $N_{ECM_{15}}$, $RelB_{20\%}$, and AUCEC together. For RQ6, we compare the findings of RQS to the results if we use the benchmark criteria from Herbold et al. [22]. Thus we determine how different the ranking from RQS is from a ranking using the metrics $AUC$, $F$-measure, $G$-measure, and MCC. The definition and reasons for selecting these metrics can be found in the benchmark by Herbold et al. [22].

We use actual defect counts for the number of true positives and false negatives to compute the above metrics. Thus, if we have an instance with two defects, it carries twice the weight for the above performance metrics. The only exception to this is RQ1, where we compare using binary labels to using defect counts. With binary labels, we do not care about the number of defects in a class and just label it as defective or non-defective, meaning that classes with five defects have the same weight as classes with one defect.

In case of ties, i.e., two instances with the same score according to a prediction model, we use the size of the instance as tie breaker and say that the smaller instances get additional quality assurance first. This tie-breaking strategy was proposed by Rahman et al. [55].

For RQ1, we compare the mean performance of each approach achieved using binary labels for calculating metric values with numerical values for calculating the metrics. We say that there is a difference if the mean value is statistically significantly different and the effect size is non-negligible. For the testing of statistical significance, we use the non-parametric Mann-Whitney-U test [42]. In case the difference is statistically significant, we measure the effect size using Cohen's $d$ [11]. According to Cohen, the effect size is negligible for $d < 0.2$, small for $0.2 \leq d < 0.5$, medium for $0.5 \leq d < 0.8$, and large for $d \geq 0.8$. We used Levene's test [5] to test if the homoscedasticity assumption of Cohen's $d$ is fulfilled. In case we find that binary labels lead to statistically significantly different results with a non-negligible effect size, data sets with binary labels instead of defect counts should not be used for cost-sensitive evaluations.

For the research questions RQ2-RQ6, we require a ranking of multiple approaches and, thus, a more complex statistical testing procedure. Our first step was to determine if we can use ANOVA [16] in combination with a Scott-Knott test [60], which is a popular choice in recent defect prediction literature that compares multiple approaches to each other, e.g., [18, 23, 63, 64]. The advantage of ANOVA and Scott-Knott is a clear and non-overlapping ranking of results. However, ANOVA has the heavy assumptions that all populations follow a normal distribution and are homoscedastic. We used the Shapiro-Wilk test [61] to test if the performance values are normally distributed and used Levene's test [5] to test for homoscedasticity. Unfortunately, both conditions are frequently broken by the data.

Therefore, we decided to use the less powerful but non-parametric Friedman test [16] with the post-hoc Nemenyi test [48] instead. The Nemenyi test compares the distances between the mean ranks of multiple pair-wise comparisons between all approaches on all products of a data set. The main drawback of this test is that the ranks between approaches may be overlapping. To deal with this issue, we follow the strategy suggested by Herbold et al. [25] to create non-overlapping groups of statistically significantly different results. Herbold et al. suggest to start with the best ranked approach and always create a new group, if the difference in ranking between two subsequently ranked approaches is greater than the critical distance. At the cost of discriminatory power of the test, this ensures that the resulting groups are non-overlapping and statistically significantly different.

Moreover, we took pattern from Tantithamthavorn et al.'s modification of the Scott-Knott test [64] and adopted the proposed effect size correction. This means that we use Cohen's $d$ to measure the effect size between two subsequently ranked groups and merge them if the effect size is negligible, i.e., $d < 0.2$.

The final step of the statistical analysis is the generation of the rankscore from the ranking. The rankscore was introduced by Herbold et al. [24] to deal with the problem of different group sizes that occur when ranks of groups of approaches are created. Not every group will have the same number of approaches. This means that the number of the group-ranking becomes a bad estimator for the performance of the group. Basically, it is a difference if you are in the second-ranked group and there is one approach in the first-ranked group or there are ten approaches in the first-ranked group. The rankscore takes care of this problem by transforming...
We now present the results of our benchmark. A replication kit rankscore
Table 3: Mean performance with binary labels and defect counts, as well as the p-value of the Mann-Whitney-U test.

| Dataset    | Metric  | Binary | Defect | p-value (d) |
|------------|---------|--------|--------|-------------|
| AEEEM      | NECM15  | 1.92   | 2.25   | 0.012       |
|            | RelB20% | 0.28   | 0.29   | 0.288       |
|            | AUCEC   | 0.56   | 0.57   | 0.236       |

| JURECZKO  | NECM15  | 3.44   | 4.10   | <0.001     |
|           | RelB20% | 0.23   | 0.21   | 0.631       |
|           | AUCEC   | 0.53   | 0.54   | 0.439       |

the ranks into a normalized representation based on the percentage of approaches that are on higher ranks, i.e.,
\[
\text{rankscore} = 1 - \frac{\text{#(approaches ranked higher)}}{\text{#(approaches)}} - 1
\]

For example, a rankscore = 1 is perfect meaning that no approach is ranked higher, a rankscore = 0.7 would mean that 30% of approaches are ranked higher.

We apply this statistical evaluation procedure to each combination of data set and performance measure. For research questions RQ2-RQ4, we then evaluate the mean rankscore on both data sets for the metrics NECM15, RelB20%, and AUCEC, respectively. For RQ5, we evaluate the mean rankscore on both data sets and for all three cost metrics. For RQ6, we evaluate the mean rankscore on both data sets for the machine learning metrics AUC, F-measure, G-measure, and MCC. To evaluate the relationship between the rankings produced with cost metrics and machine learning metrics for RQ6, we evaluate the correlation between both using Kendall’s τ [34]. Kendall’s τ is a non-parametric correlation measure between ranks of results, i.e., the ordering of results produced, which is exactly what we are interested in.

4 RESULTS

We now present the results of our benchmark. A replication kit that provides the complete source code and data required for the replication of our results, a tutorial on how to use the replication kit, as well as additional visualizations including plots for that list all classifiers and not only the best classifiers, and critical distance diagrams for the Nemenyi tests are available online.5

RQ1: Does it matter if we use defect counts or binary labels for cost metrics?

Table 3 shows the mean values for all cost metrics on the two data sets where defect counts are available. We also report the p-values determined by the Mann-Whitney-U test, and in case of significance, i.e., if p-value < 0.005, we also report the value for Cohen’s d in parenthesis. We observe statistically significant differences for NECM15 on the JURECZKO data. Levene’s test showed that the results are homoscedastic. The effect size is d = 0.52, i.e., medium.

Table 4: Mean results for NECM15 with rankscore in parenthesis.

| Approach        | JURECZKO | AEEEM     |
|-----------------|----------|-----------|
| ALL-LR          | 4.7 (0.29) | 1.31 (0.73) |
| Amasaki15-NB    | 2.9 (0.57) | 1.56 (0.54) |
| CamargoCruz09-NB| 2.92 (0.57) | 1.49 (0.54) |
| Canfora13-MODEP100 | 0.55 (1) | 0.76 (0.92) |
| FZhang16-SC     | 3.65 (0.43) | 2.35 (0.08) |
| Herbold13-NET  | 1.44 (0.86) | 0.91 (0.77) |
| Hosseini16-SBS  | 2.72 (0.57) | 1.63 (0.27) |
| Kawata15-NET    | 4.18 (0.43) | 1.61 (0.5) |
| Koshgoftaar08-LR| 3.99 (0.43) | 1.13 (0.88) |
| Liu10-GP        | 1.3 (1)  | 0.82 (1)  |
| Ma12-NET        | 3.16 (0.57) | 1.27 (0.73) |
| Menzies11-NB    | 4.48 (0.43) | 2.07 (0.12) |
| Nam15-DT        | 1.94 (0.86) | 1.11 (0.73) |
| Nam15-RF        | 1.94 (0.86) | 1.11 (0.73) |
| Panichella14-CODEP-BN  | 3.66 (0.57) | 1.45 (0.54) |
| Peters12-LR     | 4.04 (0.43) | 1.41 (0.54) |
| Peters13-LR     | 4.04 (0.43) | 1.41 (0.54) |
| Peters15-NB     | 2.88 (0.57) | 1.73 (0.42) |
| PHe15-LR        | 5.36 (0.29) | 1.21 (0.62) |
| RANDOM           | 3.84 (0.43) | 2.48 (0.08) |
| Ryu14-VCBSVM    | 2.34 (0.57) | 1.71 (0.5) |
| Ryu15-NB        | 3.11 (0.57) | 3.27 (0.04) |
| Tantitham16-NBCARET | 3.82 (0.43) | 1.97 (0.12) |
| FIX              | 0.53 (1)  | 0.72 (0.96) |
| Turhan09-LR     | 3.88 (0.43) | 1.19 (0.73) |
| Uchigaki12-LE   | 1.56 (0.86) | 3 (0.04) |
| Watanabe08-NET | 2.94 (0.57) | 1.57 (0.42) |
| YZhang15-MAXVOTE| 3.49 (0.43) | 1.34 (0.65) |
| ZHe13-NET       | 2.71 (0.57) | 0.87 (1)  |
| Zimmermann09-LR | 3.14 (0.57) | 1.57 (0.54) |

Answer RQ1: For the metric NECM15, defect counts yield significantly different results with a medium effect size in comparison to using binary labels. Therefore, only data with defect counts should be used for evaluations using NECM15. Consequently, we conclude that only data with defect counts should be used for benchmarking with cost metrics and may not use the MDP data and RELINK data used by Herbold et al. [24] for our benchmark.

RQ2: Which approach performs best if quality assurance is applied according to the prediction?

Figure 1(a) shows the best approaches ranked using NECM15 by their mean rankscore over both data sets. The mean value of NECM15 and the rankscore for each of these best approaches are listed in Table 4. The best ranking approach is Liu10-GP with a perfect mean rankscore of 1, i.e., for both data sets no approach is significantly better. The trivial baseline FIX, i.e., predicting all code as defective, is a close second with a mean rankscore of 0.981. Thus, only one approach beats the trivial baseline FIX for this performance metric. Another approach, Canfora-MODEP100 is very close to FIX with a rankscore of 0.962. If we look at the actual values of NECM15 and

5Reference to the replication kit removed due to double blind review. Will be uploaded to Zenodo in case of acceptance and evaluated through the artifact evaluation. We provide the visualizations and the statistical analysis code as supplemental material to this submission.
Figure 1: Mean rankscore over all data sets. The black diamonds depict the mean rankscore, the gray points in the background are the rankscores over which the mean is taken. We list only the result achieved with the best classifier for each approach.

| Approach | Rank | Approach | Rank |
|----------|------|----------|------|
| Liu10-GP | 1.0  | Kawata15−NET | 0.98 |
| Tantithamthavorn16−ABCARET | 0.97 | Kawata15−SVM | 0.98 |
| All−SVM | 0.95 | Liu10-GP | 0.98 |
| All−LR | 0.95 | Liu10-GP | 0.98 |
| Fix | 0.95 | Liu10-GP | 0.98 |

Table 5: Detailed results for NECM15 on the AEEEM data for Liu10-GP and FIX.

| Product | NECM15 | Rank | NECM15 | Rank |
|---------|--------|------|--------|------|
| eclipse | 0.70   | 11   | 0.68   | 9    |
| equinox | 0.84   | 5    | 0.44   | 2    |
| lucene  | 0.75   | 21   | 0.87   | 38   |
| mylyn   | 1.03   | 6    | 0.83   | 2    |
| pde     | 0.78   | 1    | 0.79   | 5    |

| mean    | 0.82   | 8.8  | 0.72   | 11.2 |

not the rankscore, FIX actually has better mean values than Liu10-GP for all three data sets, even though the ranking is worse on the AEEEM data. This anomaly is possible due to the nature of the Nemenyi test. The Nemenyi test is based on the ranking in pairwise comparisons of all approaches, not on the mean value. Table 5 shows NECM15 value and ranking of both Liu10-GP and FIX on each product in the AEEEM data. The equinox and the lucene products are most interesting. On equinox, FIX has an advantage of 0.4 over Liu10-GP with respect to the metric NECM15. However, the difference in ranks to Liu10-GP is only three. On lucene, Liu10-GP has a only an advantage of 0.12 over FIX with respect to the metric NECM15, but the difference in ranks to FIX is 27. Thus, while this anomaly seems counter intuitive, from a ranking perspective and also for the statistical test, it is correct. Such effects are the reason why checking for assumptions and choosing appropriate statistical tests is important, as these effects are due to the heteroscedacity of the data. This also shows that pure comparisons of characteristics like mean or median values are not sufficient for the ranking of multiple approaches.

Answer RQ2: Liu10-GP yields the best cost performance assuming missed defects are 15 times more expensive than additional quality assurance costs through false positive predictions. The other 25 approaches perform worse in terms of
Figure 2: Mean rankscore over all data sets for the metrics AUC, F-measure, G-measure, and MCC. The black diamonds depict the mean rankscore, the gray points in the background the rankscores over which the mean is taken. We list only the results for the classifiers that are listed in Figure 1(d), i.e., those performing best according to the metrics NECM15, RelB2015, and AUCEC.

RQ3: Which approach performs best if additional quality assurance can only be applied to a small portion of the code?

Figure 1(b) shows the best approaches ranked using RelB2015 by their mean rankscore over both data sets. The mean value of RelB2015 and the rankscore for each of these best approaches are listed in Table 6. There are actually ten approaches with a perfect rankscore of 1, i.e., Zimmermann09-SVM, Watanabe09-SVM, Turhan09-SVM, Ryu15-SVM, PHe15-SVM, Peters15-SVM, Ma12-SVM, CamargoCruz09-SVM, Amasaki15-SVM, and the trivial baseline FIX. Notably, all of the CPDP approaches use the SVM as classifier. Herbold et al. [24] found in their benchmark, that unless the bias towards non-defective instances is treated, SVMs often yield trivial or nearly trivial classifiers. We checked the raw results and found that this applies here, too. All of these SVMs are either trivial classifiers predicting only one class, or nearly trivial. Thus, they are nearly the same as the trivial baseline FIX. Since we use the size of entities as tie-breaker, this means that ranking code by size starting with the smallest instances until 20% of code is covered, is more efficient than actual CPDP. When we investigate at the actual values of RelB2015, we can conclude that we find on average 32%–35% percent of the defects that way, depending on the data set and which of the top ranked approaches is used.

Answer RQ3: Trivial or nearly trivial predictions perform best if only a small portion of the code undergoes additional quality assurance.

RQ4: Which approach performs best independent of the prediction threshold?

Figure 1(c) shows the best approaches ranked using AUCEC by their mean rankscore over both data sets. The mean value of AUCEC and the rankscore for each of these best approaches are listed in Table 7. The results are very similar to the results from RQ3, i.e., we have a large group of approaches with a perfect rankscore of 1, all of which use SVM as classifiers including the baseline FIX. Thus, starting with small code entities is again the best strategy.

Answer RQ4: Trivial or nearly trivial predictions perform best without a fix prediction threshold.

Table 6: Mean results for RelB20% with rankscore in parenthesis.

| Approach               | JURECZKO | AEEEM   |
|------------------------|----------|---------|
| ALL-SVM                | 0.33 (1) | 0.3 (0.82) |
| Amasaki15-SVM          | 0.34 (1) | 0.35 (1)   |
| CamargoCruz09-SVM      | 0.34 (1) | 0.35 (1)   |
| Canfora13-MODEP100     | 0.33 (1) | 0.32 (0.94) |
| FZhang16-SC            | 0.26 (0.8) | 0.28 (0.71) |
| Herbold13-LR           | 0.25 (0.8) | 0.36 (1)   |
| Herbold13-NET          | 0.26 (0.8) | 0.32 (1)   |
| Herbold13-SVM          | 0.24 (0.8) | 0.33 (1)   |
| Hosseini16-SBS         | 0.12 (0.2) | 0.29 (0.71) |
| Kawata15-SVM           | 0.34 (1) | 0.3 (0.82)  |
| Koshgoftaar08-SVM      | 0.34 (1) | 0.32 (0.94) |
| Liu10-GP               | 0.27 (0.8) | 0.31 (0.94) |
| Ma12-SVM               | 0.33 (1) | 0.32 (1)    |
| Menzies11-SVM          | 0.3 (0.8) | 0.31 (0.82) |
| Nam15-DT               | 0.24 (0.8) | 0.29 (0.82) |
| Nam15-RF               | 0.24 (0.8) | 0.29 (0.82) |
| Panichella14-CODEP-LR  | 0.3 (0.8) | 0.28 (0.82) |
| Peters12-SVM           | 0.33 (1) | 0.32 (0.94) |
| Peters13-SVM           | 0.33 (1) | 0.32 (0.94) |
| Peters15-SVM           | 0.34 (1) | 0.33 (1)    |
| PHe15-SVM              | 0.34 (1) | 0.32 (1)    |
| RANDOM                 | 0.26 (0.8) | 0.25 (0.35) |
| Ryu14-VCBSVM           | 0.24 (0.8) | 0.31 (0.94) |
| Ryu15-SVM              | 0.33 (1) | 0.32 (1)    |
| Tantitham.16-SVMCARET  | 0.2 (0.4) | 0.29 (0.82) |
| FIX                    | 0.34 (1) | 0.32 (1)    |
| Turhan09-SVM           | 0.32 (1) | 0.33 (1)    |
| Uchigaki12-LE          | 0.11 (0.2) | 0.2 (0.06)  |
| Watanabe08-SVM         | 0.32 (1) | 0.33 (1)    |
| YZhang15-BAG-NB        | 0.11 (0.2) | 0.28 (0.82) |
| ZHe13-LR               | 0.23 (0.8) | 0.33 (1)    |
| Zimmermann09-SVM       | 0.34 (1) | 0.33 (1)    |
Table 7: Mean results for AUCEC with rankscore in parenthesis.

| Approach             | JURECZKO | AEEEM |
|----------------------|----------|-------|
| ALL-SVM              | 0.63 (1) | 0.61 (0.95) |
| Amasaki15-SVM        | 0.63 (1) | 0.63 (1) |
| CamargoCruz09-SVM    | 0.63 (1) | 0.63 (1) |
| Canfora13-MODEP100   | 0.62 (1) | 0.61 (0.95) |
| FZhang16-SC          | 0.55 (0.6) | 0.57 (0.74) |
| Herbold13-LR         | 0.54 (0.6) | 0.63 (1) |
| Herbold13-SVM        | 0.54 (0.6) | 0.62 (1) |
| Hosseini16-SBS       | 0.48 (0.2) | 0.54 (0.63) |
| Kawata15-SVM         | 0.63 (1) | 0.61 (0.95) |
| Koshgoftaar08-SVM    | 0.63 (1) | 0.62 (1) |
| Liu10-GP             | 0.57 (0.8) | 0.6 (0.79) |
| Ma12-SVM             | 0.63 (1) | 0.61 (1) |
| Menzies11-SVM        | 0.62 (1) | 0.62 (1) |
| Nam15-SVM            | 0.55 (0.6) | 0.62 (1) |
| Panichella14-CODEP-LR| 0.6 (1) | 0.6 (0.95) |
| Peters12-SVM         | 0.63 (1) | 0.6 (0.95) |
| Peters13-SVM         | 0.63 (1) | 0.6 (0.95) |
| Peters15-SVM         | 0.62 (1) | 0.62 (1) |
| PHe15-SVM            | 0.63 (1) | 0.61 (1) |
| RANDOM                | 0.57 (0.6) | 0.55 (0.74) |
| Ryu14-VCBSVM         | 0.54 (0.6) | 0.6 (0.89) |
| Ryu15-SVM            | 0.62 (1) | 0.61 (1) |
| Tantitham16-SVMCARET | 0.55 (0.6) | 0.62 (1) |
| FIX                  | 0.63 (1) | 0.61 (1) |
| Turhan09-SVM         | 0.63 (1) | 0.61 (0.95) |
| Uchigaki12-LE        | 0.46 (0.2) | 0.49 (0.11) |
| Watanabe08-SVM       | 0.62 (1) | 0.62 (1) |
| YZhang15-MAXVOTE     | 0.53 (0.6) | 0.54 (0.42) |
| ZHe13-LR             | 0.55 (0.6) | 0.61 (1) |
| ZHe13-NB             | 0.52 (0.6) | 0.6 (1) |
| ZHe13-SVM            | 0.54 (0.6) | 0.62 (1) |
| Zimmermann09-SVM     | 0.62 (1) | 0.62 (1) |

Answer RQ5: No CPDP approach outperforms our trivial baseline on average over three performance metrics. The best performing CPDP approaches are Canfora13-MODEP100 and Liu10-GP.

RQ6: Is the overall ranking based on cost metrics different from the overall ranking based on the AUC, F-measure, G-Measure, and MCC?

Figure 2 shows the rankscores of the best ranked approaches from RQ5, but ranked with AUC, F-measure, G-measure, and MCC instead, i.e., the metrics used by Herbold et al. [24, 25] for ranking with machine learning metrics that do not consider costs. In case the rankings are correlated, we would expect that the rankscores are roughly sorted in descending order from top to bottom. However, this is clearly not the case. The two top ranking approaches with cost metrics, i.e., FIX and Canfora13-MODEP100 have both very low rankscores with the metrics used by Herbold et al., the next ranking approaches are much better with at least mediocre rankscores. Please note that the rankscore values here are not the same as determined by Herbold et al. [24, 25], because we only use two of the five data sets for this comparison. We confirmed this visual observation using Kendall’s τ as correlation measure We observe almost no correlation of $\tau = -0.047$ between both rankings.

Answer RQ6: The cost-sensitive ranking is completely different from the ranking based on AUC, F-measure, G-measure, and MCC. Thus, machine learning metrics are unsuited to predict the cost efficiency of approaches, and cost metrics are likewise unsuited to predict the performance measured with machine learning metrics.

5 DISCUSSION

Our results cast a relatively devastating light on the cost efficiency of CPDP. Based on our results, it just seems better to follow a trivial approach and assume everything as equally likely to contain defects. What we find notable is that most of the state of the art of CPDP has ignored cost metrics. Only Khoshgoftaar08, Liu10, Uchigaki12, Canfora13, Panichella14, and YZhang15, i.e., six out of 26 approaches used any effort or cost related metrics when evaluating their work. Only two of these approaches, i.e., Liu10 and Canfora13 optimize for costs. These are also the two best ranked approaches after the trivial baseline if we use all three cost metrics. Liu10 weights false negatives fifteen times stronger than false positives, i.e., puts a strong incentive on identifying defective instances in comparison to misidentifying instances as defective. This incentive means that they optimize the metric NECM15. As a results, Liu10-GP is better ranked than the trivial baseline, which is the only time this happen for all approaches and metrics in our benchmark. Canfora13 use two objectives for optimization: the recall\(^2\) and the effort in lines of code considered. This is similar to AUCEC, but not sufficient to outperform our trivial baseline even for that metric. Thus, optimizing for cost directly seems to be vital for CPDP if the created models should be cost efficient.

Another interesting aspect of our findings is the lack of correlation between the ranking using cost metrics and machine learning metrics. One would expect that good classification models in terms of machine learning metrics, also perform well under cost considerations. We believe that this correlation is missing because the performance of the CPDP models is too bad, especially the precision,\(^3\) but also in terms of recall. According to the benchmark from Herbold et al. [25], only very few predictions achieve a recall \(\geq 0.7\) and precision \(\geq 0.5\) at the same time. In other words, finding 70% of the defects with at least 50% of the predicted instances being

---

\(^2\)Percentage of predicted defective instances.

\(^3\)Percentage of defective predictions that are actually not defective.
we believe that this sample size is major threat to the validity of the benchmark's construction influences the results. Threats to the validity of our results include unsuitable choice of performance metrics for the research questions, unsuitable statistical tests, noisy or mislabeled data [31], as well as defects in our implementations. Within this paper, we present a benchmark on CPDP using effort and cost metrics. We replicated 26 approaches from the state of the art published between 2008 and 2016. Our results show that a trivial approach that predicts everything as defective performs better than the state of the art for CPDP under cost considerations. The two best CPDP were proposed by Liu et al. [40] and Canfora et al. [7, 8] and are close to the trivial predictions in performance. These are also the only two approaches, that directly optimize costs. We suspect that the generally insufficient performance of CPDP models, that was already determined in another benchmark by Herbold et al. [24], is the reason for the bad performance of CPDP in a cost-sensitive setting. It seems that optimizing directly for cost and not performance of the prediction model is currently the only way to produce relatively cost-efficient CPDP models.

In our future work, we will build on the findings of this benchmark and use the gained insights to advance the state of the art. We plan to define a general CPDP framework, that will allow a better selection of optimization criteria for approaches. We want to see if it is possible to build a wrapper around the defect prediction models that can make them optimize for costs, e.g., by injecting other performance metrics into the machine learning algorithms as optimization criteria, or manipulate the training data such that performance estimations, e.g., based on the error are more similar to actual costs. We hope to advance the state of the art this way to be more cost-efficient, such that CPDP becomes a significant improvement in comparison to assuming everything is defective. In parallel to this, we will collect more defect prediction data in order to scale up the sample size and allow better conclusions about the generalizability of defect prediction results and reduce this major threat to the validity of defect prediction research.

ACKNOWLEDGMENTS

The authors would like to thank GWGD for the access to the scientific compute cluster used for the training and evaluation of thousands of defect prediction models.

REFERENCES

[1] S. Amasaki, K. Kawata, and T. Yokogawa. 2015. Improving Cross-Project Defect Prediction Methods with Data Simplification. In Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on. 96–103. https://doi.org/10.1109/SEAA.2015.25

[2] D. J. Benjamin, J. O. Berger, M. Johannesson, B. A. Nosek, E. Wagenmakers, LotsofOtherPeople, and Dustin Tingley. 2017. Redefine Statistical Significance. Human Nature Behavior (2017).
Benchmarking cross-project defect prediction approaches with costs metrics

[3] Leo Breiman. 2001. Random Forests. Mach. Learn. 45, 1 (Oct. 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[4] David S Broomhead and David Lowe. 1988. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical Report, DTIC Document.

[5] Morton B. Brown and Alan B. Forsythe. 1974. Robust Tests for the Equality of Variances. J. Amer. Statist. Assoc. 69, 346 (1974), 364–367. http://www.jstor.org/stable/22826590

[6] Ana Erika Camargo Cruz and Koichiro Ochimizu. 2009. Towards logistic regression models for predicting fault-prone code across software projects. In Proc. 3rd Int. Symp. on Empirical Softw. Eng. and Measurement (ESEM). IEEE Computer Society. 4. https://doi.org/10.1109/ESEM.2009.5316002

[7] Gerardo Canfora, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annabale Panichella, and Sebastiano Panichella. 2013. Multi-Objective Cross-Project Defect Prediction. In Proc. 6th IEEE Int. Conf. Softw. Testing, Verification and Validation (ICST).

[8] Gerardo Canfora, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annabale Panichella, and Sebastiano Panichella. 2015. Defect prediction as a multi-objective optimization problem. Software Testing, Verification and Reliability 25, 4 (2015), 426–459. https://doi.org/10.1002/stvr.1570

[9] Rich Caruana and Alexandru Niculescu-Mizil. 2006. An Empirical Comparison of Alternative Tests of Significance for Software Defect Prediction. Empirical Software Engineering, 11, 4 (2006), 385–422. https://doi.org/10.1007/s10664-006-9058-3

[10] J. Cohen. 1988. Statistical Power Analysis for the Behavioral Sciences. Lawrence Erlbaum Associates.

[11] J. Roy Stat Soc B 29 (1958), 215–242.

[12] Marco D’Ambros, Michele Lanza, and Romain Robbes. 2010. An Extensive Comparison of Bug Prediction Approaches. In Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR). IEEE Computer Society.

[13] K. Kawata, S. Amasaki, and T. Yokogawa. 2015. Improving Relevancy Filtering Methods for Cross-Project Defect Prediction. In Applied Computing and Information Technology 2nd International Conference on Computational Science and Intelligence (ACSIT 2015), 2015 International Conference on. 2–7. https://doi.org/10.1109/ACSIT-CSI.2015.104

[14] Maurice G. Kendall. 1948. Rank correlation methods. London. http://www.gda.gov.br/db/2/1/CMD/?ACT=7ACT%28R7%29&YOP=101&TRM=ppm+18481959x8 SOURCES=flies_bibsonomy

[15] T. Menzies, A Butcher, D. Cok, A Marcus, L. Layman, F. Shull, B. Turhan, and N. Seliya. 2010. Evolutionary Optimization of Software Defect Prediction Models. In Proc. 6th Int. Conf. on Predictive Models in Softw. Eng. (PROMISE), ACM, Article 9, 10 pages. https://doi.org/10.1145/1868328.1868342

[16] Yi Liu, T.M. Khoshgoftaar, and N. Seliya. 2010. Evolutionary Optimization of Software Defect Prediction Models. In Proc. 6th Int. Conf. on Predictive Models in Softw. Eng. (PROMISE). PROMISE, ACM, Article 9, 10 pages. https://doi.org/10.1145/1868328.1868342

[17] Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang, Can Candan, and Tyler Hunt. 2016. caret: Classification and Regression Training. https://CRAN.R-project.org/package=caret R package version 6.0-73.

[18] Yi Liu, T.M. Khoshgoftaar, and N. Seliya. 2010. Evolutionary Optimization of Software Defect Prediction Models. In Proc. 6th Int. Conf. on Predictive Models in Softw. Eng. (PROMISE). PROMISE, ACM, Article 9, 10 pages. https://doi.org/10.1145/1868328.1868342

[19] K. Herzig, S. Just, A. Rau, and A. Zeller. 2013. Predicting defects using change geoneties. In Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on. 118–127. https://doi.org/10.1109/ISSRE.2013.6698911
Jaechang Nam, S.J. Pan, and Sung hun Kim. 2013. Transfer defect learning. In Software Engineering (ICSE), 2013 35th International Conference on. 382–391. https://doi.org/10.1109/ICSE.2013.6605584

P.B. Nemenyi. 1963. Distribution-free Multiple Comparison. Ph.D. Dissertation. Princeton University.

Sinno Jialin Pan, I.W. Tsang, J.T. Kwok, and Qiang Yang. 2011. Domain Adaptation. In Empirical Software Engineering (ICSE), 2011 33rd International Conference on. 396–405. https://doi.org/10.1109/ICSE.2011.60

Duksan Ryu, Okjoo Choi, and Jongmoon Baik. 2014. Value-cognitive boosting approach for cross-project defect prediction. In Software Quality Journal. 54, 1 (2015), 969–980. https://doi.org/10.1007/s11390-015-1575-5

Chakirrat Tangtithamthavorn, Shame McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2016. Automated Parameter Optimization of Classification Techniques for Defect Prediction Models. In Proceedings of the 38th International Conference on Software Engineering. ACM. https://doi.org/10.1145/2884781.2884857

Chakirrat Tangtithamthavorn, Shame McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2017. An Empirical Comparison of Model Validation Techniques for Defect Prediction Models. IEEE Transactions on Software Engineering 43, 1 (2017), 1–18. https://doi.org/10.1109/TSE.2016.2584050

Fabian Trautsch, Steffen Herbold, Phillip Markedouski, and Jens Grabowski. 2016. Adressing Problems with External Validity of Repository Mining Studies Through a Smart Data Platform. In Proceedings of the 13th International Conference on Mining Software Repositories (MSR ’16). ACM, New York, NY, USA, 97–108. https://doi.org/10.1145/2901739.2901753

Fabian Trautsch, Steffen Herbold, Philip Markedouski, and Jens Grabowski. 2017. Addressing problems with replicability and validity of repository mining studies through a smart data platform. Empirical Software Engineering (08 Aug 2017). https://doi.org/10.1007/s10664-017-9537-x

Burak Turhan, Tim Menzies, AyşeB. Bener, and Justin Di Stefano. 2009. On the relative value of cross-company and within-company data for defect prediction. Empirical Software Eng. 14 (2009), 540–578. Issue 5. https://doi.org/10.1007/s10664-008-9103-7

Burak Turhan, Ayse Tosun Musiril, and Ayse Bener. 2013. Empirical evaluation of the effects of mixed project data on learning defect predictors. Information & Software Technology 55, 6 (2013), 1101–1118. https://doi.org/10.1016/j.infsof.2012.10.003

B. Turhan, A. Tosun, and A. Bener. 2011. Empirical Evaluation of Mixed-Project Defect Prediction Models. In Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on. 396–403. https://doi.org/10.1109/SEAA.2011.59

S. Uchigaki, S. Uchida, K. Toda, and A. Monden. 2012. An Ensemble Approach of Simple Regression Models to Cross-Project Fault Prediction. In Software Engineering, Artificial Intelligence, Networking and Parallel Distributed Computing (SNPD), 2012 18th ACIS International Conference on. 476–481. https://doi.org/10.1109/SNPD.2012.34

F. Peters, T. Menzies, and L. Layman. 2015. LACE2: Better Privacy-Preserving Data Sharing for Cross Project Defect Prediction. In Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on, Vol. 1. 801–811. https://doi.org/10.1109/ICSE.2015.92

F. Peters, T. Menzies, and L. Layman. 2015. LACE2: Better Privacy-Preserving Data Sharing for Cross Project Defect Prediction. In Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on, Vol. 1. 801–811. https://doi.org/10.1109/ICSE.2015.92

Joel R. Wieland, Ayse Tosun, and Ayse Bener. 2016. Transfer Learning Using a Connectivity-based Unsupervised Classifier. In Proceedings of the 13th Joint Meet. Eur. Softw. Eng. Conf. (ESEC) and the ACM SIGSOFT Symp. Found. Softw. Eng. (FSE). ACM, 91–100. https://doi.org/10.1145/2901739.2901753