A Benchmark Study on the Effectiveness of Search-based Data Selection and Feature Selection for Cross Project Defect Prediction

Seyedrebvar Hosseini[a], Burak Turhan[b], Mika Mäntylä[a]

[a] *M3S, Faculty of Information Technology and Electrical Engineering*
University of Oulu, 90014, Oulu, Finland
[b] *Department of Computer Science*
Brunel University London, UB8 3PH, United Kingdom

Abstract

Context: Previous studies have shown that steered training data or dataset selection can lead to better performance for cross project defect prediction(CPDP). On the other hand, feature selection and data quality are issues to consider in CPDP.

Objective: We aim at utilizing the Nearest Neighbor (NN)-Filter, embedded in genetic algorithm to produce validation sets for generating evolving training datasets to tackle CPDP while accounting for potential noise in defect labels. We also investigate the impact of using different feature sets.

Method: We extend our proposed approach, Genetic Instance Selection (GIS), by incorporating feature selection in its setting. We use 41 releases of 11 multi-version projects to assess the performance GIS in comparison with benchmark CPDP (NN-filter and Naive-CPDP) and within project (Cross-Validation(CV) and Previous Releases(PR)). To assess the impact of feature sets, we use two sets of features, SCM+OO+LOC(all) and CK+LOC(ckloc) as well as iterative info-gain subsetting(IG) for feature selection.

Results: GIS variant with info gain feature selection is significantly better than NN-Filter (all,ckloc,IG) in terms of F1 ($p = values \ll 0.001$, Cohen's $d = \{0.621, 0.845, 0.762\}$) and G ($p = values \ll 0.001$, Cohen's $d = \{0.899, 1.114, 1.056\}$), and Naive CPDP (all,ckloc,IG) in terms of F1 ($p = values \ll 0.001$, Cohen's $d = \{0.743, 0.865, 0.789\}$) and G ($p =$

Email addresses: `rebvar@oulu.fi` (Seyedrebvar Hosseini), `burak.turhan@brunel.ac.uk` (Burak Turhan), `mika.mantyla@oulu.fi` (Mika Mäntylä)

Preprint submitted to Journal of Information and Software Technology *June 16, 2017*

values $\ll 0.001$, Cohen's $d = \{1.027, 1.119, 1.050\}$). Overall, the performance of GIS is comparable to that of within project defect prediction (WPDP) benchmarks, i.e. CV and PR. In terms of multiple comparisons test, all variants of GIS belong to the top ranking group of approaches.

Conclusions: We conclude that datasets obtained from search based approaches combined with feature selection techniques is a promising way to tackle CPDP. Especially, the performance comparison with the within project scenario encourages further investigation of our approach. However, the performance of GIS is based on high recall in the expense of a loss in precision. Using different optimization goals, utilizing other validation datasets and other feature selection techniques are possible future directions to investigate.

Keywords: Cross Project Defect Prediction, Search Based Optimization, Genetic Algorithms, Instance Selection, Training Data Selection

1. Introduction

Despite the extensive body of studies and its long history, software defect prediction is still a challenging problem in the field of software engineering [1]. Software teams mainly use software testing as their primary method of detecting and preventing defects in the development stage. On the other hand, software testing can be very time consuming while the resources for such tasks might be limited and hence, detecting defects in an automated way can save lots of time and effort [2, 3, 4].

Defect data from previous versions of the same project could be used to detect defect prone units in new releases. Prediction based on the historical data collected from the same project is called within project defect prediction (WPDP) and has been studied extensively [5, 6, 7, 8, 9, 10, 11]. New code and releases in the same project usually share many common characteristics that make them a good match for constructing prediction models. But this approach is subject to criticism as within project data is usually not available for new projects. Moreover, this problem only tends to increase as the need for software based platforms and services is growing at a rapid pace. On the other hand, there are plenty of relevant public datasets available, especially in the open source repositories [12] that can act as candidates to identify and prevent bugs in the absence and even presence of within project data. Using the available public datasets, one can investigate the usefulness of models created on the data from other projects, especially for those with limited or no defect data repository [3, 4, 13].

Various studies have focused on different aspects of defect prediction including data manipulation approaches (such as re-sampling [14, 15, 16, 17], re-weighting [17, 18, 19, 20], filtering [4, 13, 21, 22], etc.), learning technique optimization (boosting [14, 15, 17, 23], bagging[23], ensembles [22, 23, 24, 25], etc.) and metrics (feature selection [23, 26, 27, 28], different set of metrics [29]) to mention some. Predictions usually target the effectiveness of the approaches in two main categories: binary and continuous/multi-class predictions[1]. The features also come in different levels of code including function/method, class, file and even component levels [1]. As would be discussed in later sections, the experiments in this study are class level binary predictions performed on 41 Java projects. A summary of the related studies would be presented in Section 2.

Learning approach and the training data are two of the major elements in building high performance prediction models. Finding a suitable dataset (instances and features) with similar defect distribution characteristics as the test set is likely to increase the performance of prediction models [30]. Since defect detection and label assignment is based on mining version control systems [31, 32, 33], the process could be prone to errors and data quality can be questionable [34, 35]. In other words, the labels of some of the instances might not have been identified correctly, two or more instances with the same measurements can have different labels, or undetected defects may not be captured in the dataset in the time of dataset compilation.

In this study, we address these problem with a search based instance selection approach, where a mutation operation is designed to account for data quality. Using the genetic algorithm, we guide the instance selection process with the aim of convergence to datasets that match the characteristics of the test set more precisely. The fitness function at each generation is evaluated on a validation set generated via (NN)-Filter. With these, we handle the potential noise in data while tackling the training data instance selection problem with GIS.

In our earlier work [35], we combined training data instance selection with search based methods to address the potential defect labeling errors and we compared our method's performance with benchmark CPDP and WPDP (cross validation) approaches. This study not only acts as a replication of our original study but also extends it by the following ways:

- Including a more comprehensive set of datasets. In the original study, only the last available dataset from multi-version software projects were used in our experiments. Overall, 13 datasets from PROMISE repository were used to assess the performance of our proposed approach. We extend the number of datasets to 41 from 11 projects. The reason for the choosing these datasets is their maturity (multiple versions) and the choice of WPDP previous releases benchmark.

- Including an extra within project benchmark (previous releases). We used a single WPDP benchmark in our original study namely, 10 fold stratified cross validation. Since multiple releases are available for each project in this extension, we compare the performance of GIS with this WPDP benchmark as well.

- Investigating the effect of different sets of metrics and the sensitivity of GIS toward them. Chidamber & Kemerer (CK)+LOC was used in the original study, due to its reported good performances by Hall et al.[1] as well as recent CPDP studies [25, 36]. To have a better understanding of how GIS reacts to the choice of metrics we used all features and one feature selection technique, i.e. iterative infogain subsetting. We conducted these sets of experiments to address one of the threats to the validity of our original conclusions, i.e. the selection of software metrics. Interestingly, adding SCM to CK+LOC has a positive effect on the WPDP performance despite its negative effect on GIS, therefore making it necessary to be included as failing to so, would be a threat to the conclusion validity.

- Presenting a more extensive analysis, related work and discussions. A wider range of datasets, benchmarks and results, requires more comprehensive analysis. The results are presented through multiple types of diagrams (violin plots, critical difference (CD) diagrams, line plots) and the approaches are compared with different targets in mind through statistical tests for pairwise and multiple comparisons for exploring different perspectives of the achieved results.

Accordingly, the aim of this study is to answer the following research questions:

RQ1: How is the performance of GIS compared with benchmark cross project defect prediction approaches?

RQ2: How is the performance of GIS compared with the within project defect prediction approaches?

RQ3: How different feature sets affect the performance of GIS?

This paper is organized as follows: The next section summarizes the related studies on CPDP and briefly describes how our study differs. Proposed approach, datasets and experimental procedures are presented in Section 3. Section 4 presents the results of our analysis and discussions. Section 5 addresses some of the concerns that arise during the analysis and wraps up our findings. Section 6 discusses the threats to the validity of our study. Finally, the last section concludes the paper with a summary of the findings as well as directions for future work.

2. Related Work

Cross project defect prediction (CPDP) has drawn a great deal of interest recently. To predict defects in projects without sufficient training data, many researchers attempted to build novel and competitive CPDP models [4, 13, 18, 21, 31, 37, 38]. However, not all studies report good performances of CPDP [4, 16, 38].

In a series of experiments, Turhan et al. [4] observed that CPDP underperforms WPDP. They also found that despite its good probability of detection rates, CPDP causes excessive false alarms. They proposed to use (NN)-Filter to select the most relevant training data instances based on a similarity measure. Through this method, they were able to lower the high false alarm rates dramatically, but their model performance was still lower than WPDP.

Zimmermann et al. [38] performed a large scale set of CPDP experiments by creating 622 pair-wise prediction models on 28 datasets from 12 projects (open source and commercial) and observed only 21 pairs (3.4%) that match their performance criteria (precision, recall and accuracy, all greater than 0.75). This observation suggests that the majority of predictions will probably fail if training data is not selected carefully. They also found that CPDP is not symmetrical as data from Firefox can predict Internet Explorer defects, but the opposite does not hold. They argued that characteristics of data and process are crucial factors for CPDP.

He et al. [13] proposed to use the distributional characteristics (median, mean, variance, standard deviation, skewness, quantiles, etc.) for training dataset selection. They concluded that in the best cases cross project data may provide acceptable prediction results. They also state that training data from the same project does not always lead to better predictions and carefully selected cross project data may provide better prediction results than within-project (WP) data. They also found that data distributional characteristics are informative for training data selection. They used a metalearner built on top of the prediction results of the decision table learner to predict the outcome of the models before making actual predictions.

Herbold [18] proposed distance-based strategies for the selection of training data based on distributional characteristics of the available data. They presented two strategies based on EM (Expectation Maximization) clustering and NN (Nearest Neighbor) algorithm with distributional characteristics as the decision strategy. They evaluated the strategies in a large case study with 44 versions of 14 software projects and they observed that i) weights can be used to successfully deal with biased data and ii) the training data selection provides a significant improvement in the success rate and recall of defect detection. However, their overall success rate was still too low for the practical application of CPDP.

Turhan et al. [21] evaluated the effects of mixed project data on predictions. They tested whether mixed WP and CP data improves the prediction performances. They performed their experiments on 73 versions of 41 projects using Naïve Bayes classifier. They concluded that the mixed project data would significantly improve the performance of the defect predictors.

Zhang et al [39] created a universal defect prediction model from a large pool of 1,385 projects with the aim of relieving the need to build prediction models for individual projects. They approached the problem of variations in the distributions by clustering and rank transformation using the similarities among the projects. Based on their results, their model obtained prediction performance comparable to the WP models when applied to five external projects and performed similarly among projects with different context factors.

Ryu et al. [22] presented a Hybrid Instance Selection with the Nearest Neighbor (HISNN) method using a hybrid classification to address the class imbalance for CPDP. Their approach used a combination of the Nearest Neighbour algorithm and Naïve Bayes learner to address the instance selection problem.

He et al. [26] considered CPDP from the viewpoint of metrics and features by investigating the usefulness of simplified metric sets. They used a greedy approach to filter the list of available metrics and proposed to use different sets of metrics according to the defined criteria. They observed that minimum feature subsets and TOPK metrics could provide acceptable results compared with their benchmarks. They further concluded that the minimum feature subset can improve the predictions despite the acceptable loss of precision.

Feature selection and more specifically, feature matching was studied by Nam et al.[28]. Their proposed approach provided the ability of performing predictions on training and test datasets with different sets of metrics. They used statistical procedures for the feature matching processes and observed that their CPDP approach outperforms WPDP in 68% of the predictions.

While the above-mentioned studies focus on the dataset, instance and feature selection problems, none of them are using the search based approach. One such approach in defect prediction context has been considered by Liu et al., who tried to come up with mathematical expressions as their solutions that maximize the effectiveness of their approach [36]. They compared their approach with 17 non-evolutionary machine learning algorithms and concluded that the search-based models decrease the misclassification rate consistently compared with the non-search-based models.

Canfora et al. proposed a search based multi-objective optimization approach [40] for CPDP. Using multi-objective genetic algorithm NSGA-II, they tried to come up with an optimal cost effectiveness model for CPDP. They concluded that their approach outperforms the single objective, trivial and local prediction approaches. Recently, Xia et al. [41] have conducted a search based experiment consisting of a genetic algorithm phase and an ensemble phase. They utilized logistic regression as their base learner and small chunks of within project data in their settings. They compared their proposed approach, i.e. HYDRA with some of the most recent CPDP approaches and observed that it outperformed the benchmarks significantly. Even though these studies use a search based approach, they are neither focused on the instance/dataset selection nor on the data quality problem and hence, differ from our approach.

3. Research Methodology

This section describes the details of our study starting with a discussion of our motivation. We then present the proposed approach as well as the benchmark methods, datasets and metrics, and the performance evaluation criteria used in our study.

3.1. Motivation

If selected carefully, a dataset from other projects can provide a better predicting power than WP data [13] as the large pool of the available CP data has the potential to cover a larger range of the feature space. This may lead to a better match between training and test datasets and consequently to better predictions.

One of the first attempts in this area was the idea of filtering the training dataset instances [4]. In this approach, the most similar instances from a large pool containing all the training instances from other projects are selected using k-NN algorithm. Since these instances are closer to the test set based on a particular distance measure, they could potentially lead to better predictions. Using the distributional characteristics of the test and training datasets is another approach used in multiple studies [13, 42]. Clustering the instances is yet another approach used in other studies [18, 31, 39]. While these methods have been shown to be useful, the search based approach to selection is not considered by any of these papers. An evolving dataset starting with the initial datasets generated using one or a combination of these approaches can be a good candidate for a search based data selection problem.

Data Quality. Another inspiration for this work is the fact that the public datasets are prone to quality issues and contain noisy data [34, 35, 43]. Since defects are discovered over time, certain defects might not have been discovered at the time of compiling the datasets and hence, some of the instances in the training set may be misrepresenting themselves as non-defective, while with similar kind of measurements defects can exist in the test set. In contrast, while some test instances are not defective, the most similar items in the training set might be labeled as defective. In short, some of the instances in the test set can have similar measurements with the training set, yet different class labels. Please note that mislabeling may not be the only reason for such situations, and they can occur naturally, i.e. the class labels of similar measurements can differ depending on the metric set used. The acknowledgment of noise in the data and guiding the learning algorithm to account for that can lead to better predictions, as we proposed in our original paper [35] and validate it in this study.

Features. We used CK+LOC metric set originally to assess the performance of GIS and the other benchmarks. Hall et al. [1] asserted that OO (Object-Oriented) and LOC have acceptable prediction power and they

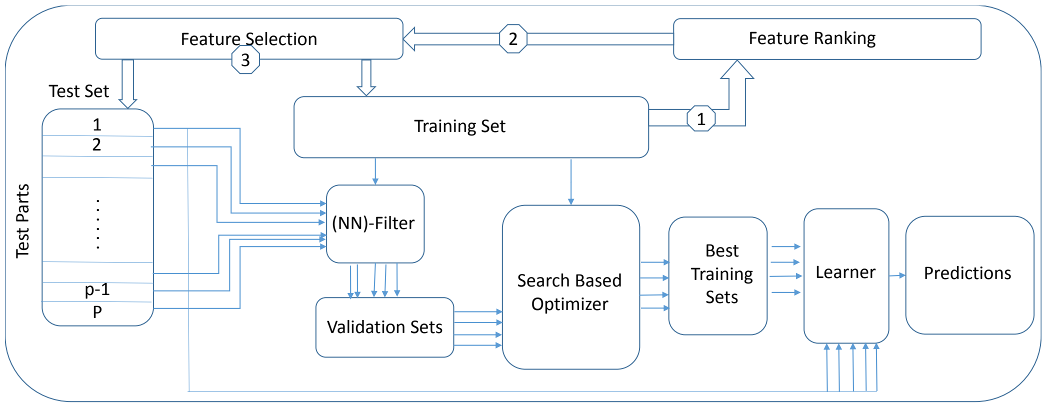


Figure 1: Summary of the search based training data instance selection and performance evaluation process used in this paper.

outperform SCM (static code metrics). Moreover, they observed that adding SCM to OO and LOC is not related to better performance [1]. Moreover, the usefulness of CK+LOC was validated in multiple studies in the context of CPDP [25, 37, 40] of which [40] involves using search based approaches. We extend our work not only by considering CK+LOC but also OO+SCM+LOC as well as feature selection.

We used the information gain concept to select the top ranked features which explain the highest entropy in the data. Information gain method is relatively fast and the ranking process does not need any actual predictions. Please note that there certainly exist more sophisticated and powerful feature selection approaches that could be used in the context of CPDP. The use of information gain feature subsetting is due to its simplicity and speed and a proof of concept that even simple refinement of the data through a guided procedure can lead to practical improvements.

Infogain is defined as follows: For an attribute A and a class C, the entropy of the class before and after observing A are as follows:

$$H(C) = -\sum_{c \in C} p(c) \log_2 p(c) \quad (1)$$

$$H(C|A) = -\sum_{a \in A} p(a) \sum_{c \in C} p(c|a) \log_2 p(c|a) \quad (2)$$

The amount of explained entropy by including A reflects the additional information acquired and is called information gain. Using this approach the

Algorithm 1 Pseudo code for GIS

```
Set numGens = The number of generations of each genetic optimization run.
Set popSize = The size of the population.
Set DATA = 41 releases from {Ant, Camel, ivy, jedit, log4j, lucene, poi, synapse, velocity, xalan,xerces}
Set FEATURES = {CKLOC, All, IG}

for FSET in FEATURES do
    for RELEASE in DATA do
        set TEST = Load Instances from RELEASE with metric set FSET
        set TRAIN = Instances from all other projects with metric set FSET
        tdSize = 0.02 * Number of instances in TRAIN
        for i = 1 to 30 do
            Set TestParts = Split TEST instances into p parts

            for each testPart in TestParts do
                Set vSet = Generate a validation dataset using (NN)-Filter method (with three distance measures).
                Set TrainDataSets = Create popSize dataset from TRAIN with replacement each with tdSize instances

                for each td in TrainDataSets do
                    Evaluate td on vSet and add it to the initial generation
                end for

                for g in range(numGens) do
                    Create a new generation using the defined Crossover and Mutation function and Elites from the
                    curent generation.
                    Combine the two generations and extract a new generation
                end for

                Set bestDS = Select the top dataset from the GA's last iteration.
                Evaluate bestDS on testPart and append the results to the pool of results.
            end for

            Calculate Precision, Recall and F1 and G from the predictions.
        end for
        Report the median of all 30 experiments
    end for
end for
```

features of the datasets are ranked from the highest to the lowest amount of entropy explained. We used iterative InfoGain subsetting [44] to select the appropriate set of features for our experiments. Iterative InfoGain subsetting starts by training the predictors using the top n ranked attributes for $n \in \{1, 2, ...\}$ and continues until a point that having $j + 1$ attributes instead of j does not improve the predictions. An improvement in predictions was measured using F1 values achieved from a 1×10 fold cross validation on the training dataset. The train test splits were identical when adding the features iteratively during the feature selection operation.

3.2. Proposed Approach

Figure 1 visualizes the whole research process reported in this paper. The details of the search based optimizer are not present in the figure and instead, they are provided in Algorithm 1 and discussed below.

The process starts with splitting the test set into p parts randomly ($p = 5$

in our experiments). Partitioning the test set into smaller chunks plays an important role in the overall procedure. By creating smaller chunks, the process of optimizing and adjusting the dataset is easier as there are less elements to consider and the datasets generated could be better representatives for these smaller chunks than the whole dataset. This procedure however, adds extra complexity to the model and the run-time would increase consequently since a search based optimizer is required for each part.

Each part (without the labels) is fed into the (NN)-Filter instance selection method in order to select the most relevant instances from the training set for the purpose of reserving a validation set, on which we optimize the search process. Please note that the training set is a combination of all the instances from other projects. We used the closest three instances with multiple distance measures to account for the possible error in using a specific distance measure. The unique instances from the generated set were selected to act as the validation dataset used to guide our instance selection process. The availability of mixed data as used in [17, 21, 41] could also potentially act as a replacement for the aforementioned similarity measures and boost the performance of our approach.

The process then randomly creates an initial population containing *popSize* datasets (*popSize*=30 in our experiments). Each population element is a dataset selected randomly and with replacement from the large pool of training set instances (see Table 1). The selected number of population members and their sizes lead to an average of 94.99% coverage (std=0.031) of the instances in the large pool of available training instances (multiple copies for some) for each iteration.

Each population member is then evaluated on the validation set, which is acquired via the (NN)-Filter in the previous step. Then, for *numGens* generations, a new population is generated and the top elements are selected to survive and move to the next generation. There is an alternative stopping criterion for GIS (described below). These procedures are repeated 30 times to address the randomness introduced by both the dataset selection and genetic operations. Below, the genetic operations and parameters are discussed in more details:

Initial Population: The initial population is generated using the random instance selection process with replacement from a large pool of instances containing all elements from other projects than the test project. The instances might contain elements included in the validation dataset as

	$\mathbf{F_1}$	$\mathbf{F_2}$	$\cdots$	$\mathbf{F_{m\text{-}2}}$	$\mathbf{F_{m\text{-}1}}$	$\mathbf{F_m}$	**L**
$\mathbf{C_1}$	6	0	$\cdots$	0	1	1	0
$\mathbf{C_2}$	3	0.97	$\cdots$	12	3	1	1
$\mathbf{C_3}$	5	0.69	$\cdots$	12.6	4	1.4	0
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\mathbf{C_{n\text{-}2}}$	3	0.98	$\cdots$	16	2	1	0
$\mathbf{C_{n\text{-}1}}$	3	0.82	$\cdots$	8.33	1	0.67	0
$\mathbf{C_n}$	16	0.73	$\cdots$	28.3	9	1.56	1

Table 1: Chromosome structure

they are not removed from the large pool of candidate training instances due to their possible usefulness for the learning process. The selection process consumes 94.99% of the initial training data on average and eliminates a group of them with each passing generation.

Chromosome Representation: Each chromosome contains a number of instances from a list of projects. A chromosome is a dataset sampled from the large training dataset randomly and with replacement. A typical chromosome example can be seen in Table 1. $\mathbf{F_i}$ represents i^{th} selected feature and **L** represents the class label. Atypical chromosome contains n instances denoted by $\mathbf{C_1}$ to $\mathbf{C_n}$.

We used a fixed size chromosome in our experiments. The size of each chromosome (dataset) was set to 0.02% of the large pool of training data from other projects. The fixed size chromosome was selected to show the effectiveness of our proposed approach with respect to the small candidate training sets generated. One might find the varying size chromosome more useful in practice as the candidates in this version might be able to capture more properties of the test set subject to prediction.

Selection: The Tournament selection is used as the selection operator of GIS. Since the population size is small in our experiments, tournament size was set to two.

Elites: A proportion of the population is moved to the next generation; those that provide the best fitness values. We transfer two of the top parents to the next generation.

Stopping Criteria: We place two limitations on the number of iterations that the genetic algorithm could progress. The first one is the maximum number of generations allowed. In this case, this number was set to 20. The reason for selecting a relatively small number of generations (20) is due to having small population sizes. The small populations was selected for the runtime considerations. Despite their sizes however, they cover 94.99% of the original training instances on average in every iteration, when creating the initial populations. Further, we observed that the process converges quickly and hence, making 20 an acceptable maximum number of generations. The other stopping criterion is the amount of benefit gained from the population generated. If the difference between the mean fitness of two consecutive populations is less than $\epsilon = 0.0001$, the genetic algorithm stops.The mentioned epsilon was selected arbitrarily to be a small number. One can expect to achieve better results by tuning these parameters.

Fitness Function: F1 * G is used as the fitness value of each population element. Each population element (a dataset) is evaluated on the validation set and fitness value is assigned to it. The selection of this fitness function is not random as both of these values (F1 and G) measure the balance between precision and recall, but in different ways.

Mutation: The mutation function handles potential data quality (e.g, noise, mislabelling etc.) issues. Randomly changing the class value of the instances from non defective to defective (and vice versa), the mutation operation guides the process through multiple generations for yielding more similar datasets. This could to some extent account for both undiscovered bugs as well as contradictory labels for similar measurements in different projects (training and test data). With the probability of $mProb = 0.05$, a number of training set instances are mutated by flipping the labels (defective $\rightarrow$ non defective or non defective $\rightarrow$ defective). Note that since the datasets could contain repetitions of an element (from the initial population generation and later from the crossover operation), if an instance is mutated, all of its repetitions are also mutated. This way, we could avoid conflicts between the items in the same dataset. The mutation process is described in Algorithm 2 formally.

Crossover: The generated training datasets used in the population could possibly have large sizes. The time for training a learner with a large dataset and validating it on a medium size validation set increases, if the size of the train and validation datasets increase. To avoid having very large datasets

Algorithm 2 Mutation

```
Input → DS: a dataset
Output → A dataset with possible mutated items

set mProb = p // Mutation probability
set mCount = c // Number of instances to mutate
set r = Random value between 0 and 1

if r < mProb then
    for i in range(mCount) do
        Randomly select an instance that is not been mutated in the same round
        Find all repeats of the same item and flip their labels
    end for

end if
```

Algorithm 3 One point crossover

```
Input → DS1 and DS2
Output → Two new datasets generated from DS1 and DS2

Set nDS1 = Empty dataset
Set nDS2 = Empty dataset
Set point = Random in the range of either of DS1 or DS2
SHUFFLE DS1 and DS2

for i = 1 to point do
    Append DS1(i) to nDS1
    Append DS2(i) to nDS2
end for

for i = point+1 to DS1's length do
    Append DS1(i) to nDS2
    Append DS2(i) to nDS1
end for

for each unique instance in nDS1 and nDS2 do
    Use the majority voting to decide the label of the instance and its repetitions.
end for
```

one point crossover was used during the crossover operation. Nevertheless, some might find two point cross over more useful. As mentioned earlier, the chromosomes are a list of instances from the large training set, selected randomly and with replacement with a fixed size. Hence one point cross over would not increase the size of the datasets when combining them. Since the chromosomes possibly contain the repetitions of one item and the mutation operation changes the label of an instance, conflicts might occur in the chromosomes generated from combining the two selected parents. In the case of conflicts, the majority voting is used to select the label of such instances. Algorithm 3 provides the pseudo-code for crossover operation.

3.3. Benchmark Methods

To have a better insight into the performance achieved by GIS, it is compared to the following benchmarks:

(NN)-Filter (CPDP): In this approach, the most relevant training instances are selected based on a distance measure [4]. In this case, we used 10 nearest neighbours and Euclidean distance. This value is similar to that utilized by multiple previous studies [4, 14, 18, 45]. The k=10 was the value of choice in the study by Turhan et al. [4] which proposed NN-Filter. The selection of k as 10 was followed by later studies such as He et al. [45] and Chen et al. [14], both of which focus on CPDP. Another CPDP study by Herbold [18] used different values of k $\in \{3, 5, 10, 15, 20, 25, 30\}$ and observed the best results for larger k values. The simplicity of the method and the comprehensive number of studies that have tested the approach are the reasons for choosing this method as a benchmark [4, 17, 22]. Also GIS uses (NN)-Filter to select the validation dataset and a benchmark is required to measure the performance difference between (NN)-Filter and GIS.

Naive (CPDP): In this approach, the whole training set is fed into the learner and the model is trained with all the training data points. This method has also been tested in many studies and provides a baseline for the comparisons [4, 17, 22]. The approach is easy and at the same time demonstrates that while the availability of large pools of data could be useful, not all the data items are.

10-Fold cross validation (WPDP): In this benchmark, we perform stratified cross validation on the test set. Many studies have reported the good or at least better performance of this approach compared with that of cross project methods [4]. Outperforming and improving WPDP is the main goals of many such studies. We refer to this benchmark as **CV** throughout our analysis.

Previous Releases (WPDP): Previous releases of the same project are used to train the prediction model. Similar to 10-fold cross validation, a good performance of this approach is expected in comparison with that of cross project methods as these older releases are more similar to the test set in comparison with datasets from other projects. More importantly, there is a higher possibility of finding even identical classes in the old and new releases of a project. Previous releases are another target of the CPDP studies as acquiring such data is still difficult in some cases. Note that the first release of each project does not have a previous release and therefore no prediction could be performed for it in this category. We use the 10 fold cross validation

Table 2: Utilized datasets and their properties

Dataset	#Classes	#DP	DP%	#LOC	Dataset	#Classes	#DP	DP%	#LOC
ant-1.3	125	20	16	37699	**lucene-2.0**	195	91	46.7	50596
ant-1.4	178	40	22.5	54195	**lucene-2.2**	247	144	58.3	63571
ant-1.5	293	32	10.9	87047	**lucene-2.4**	340	203	59.7	102859
ant-1.6	351	92	26.2	113246	**poi-1.5**	237	141	59.5	55428
ant-1.7	745	166	22.3	208653	**poi-2.0**	314	37	11.8	93171
camel-1.0	339	13	3.8	33721	**poi-2.5**	385	248	64.4	119731
camel-1.2	608	216	35.5	66302	**poi-3.0**	442	281	63.6	129327
camel-1.4	872	145	16.6	98080	**synapse-1.0**	157	16	10.2	28806
camel-1.6	965	188	19.5	113055	**synapse-1.1**	222	60	27	42302
ivy-1.1	111	63	56.8	27292	**synapse-1.2**	256	86	33.6	53500
ivy-1.4	241	16	6.6	59286	**velocity-1.4**	196	147	75	51713
ivy-2.0	352	40	11.4	87769	**velocity-1.5**	214	142	66.4	53141
jedit-3.2	272	90	33.1	128883	**velocity-1.6**	229	78	34.1	57012
jedit-4.0	306	75	24.5	144803	**xalan-2.4**	723	110	15.2	225088
jedit-4.1	312	79	25.3	153087	**xalan-2.5**	803	387	48.2	304860
jedit-4.2	367	48	13.1	170683	**xalan-2.6**	885	411	46.4	411737
jedit-4.3	492	11	2.2	202363	**xalan-2.7**	909	898	98.8	428555
log4j-1.0	135	34	25.2	21549	**xerces-1.2**	440	71	16.1	159254
log4j-1.1	109	37	33.9	19938	**xerces-1.3**	453	69	15.2	167095
log4j-1.2	205	189	92.2	38191	**xerces-1.4**	588	437	74.3	141180
					xerces-init	162	77	47.5	90718

result for the first release of each project in order to make the comparisons easier. We denote this benchmark by **PR** in the following.

Feature Selection: Each of the aforementioned benchmarks are trained and tested using three different sets of features. CK+LOC, used in our original study [35] as well as the whole set of features in the datasets which consist of OO+SCM+LOC are considered for all benchmarks. Beside these feature sets, a portion of the features ranked based on their respective information gain are used to prepare another set of benchmarks. We used iterative InfoGain subsetting method to select the appropriate features for each benchmark.

The first two benchmarks (CPDP) are used to answer RQ1 and the latter are utilized to answer RQ2. The results of different versions of GIS would be used to answer the last research question, i.e. RQ3. Each experiment is repeated 30 times to address the randomness introduced by CV and GIS.

3.4. Datasets and Metrics

We used 41 releases of 11 projects from the PROMISE repository for our experiments. These projects are open source and all of them have multiple versions. Due to the inclusion of the multi version WPDP benchmark, we skipped the use of datasets with a single version. The datasets are collected by Jureczko, Madeyski and Spinellis [31, 32]. The list of the datasets is presented in Table 2 with the corresponding size and defect information.

Table 3: List of the metrics used in this study

ID	Variable	Description
1	WMC	Weighted Methods per Class
2	DIT	Depth of Inheritance Tree
3	NOC	Number of Children
4	CBO	Coupling between Object classes
5	RFC	Response for a Class
6	LCOM	Lack of Cohesion in Methods
7	CA	Afferent Couplings
8	CE	Efferent Couplings
9	NPM	Number of Public Methods
10	LCOM3	Normalized version of LCOM
11	LOC	Lines Of Code
12	DAM	Data Access Metric
13	MOA	Measure Of Aggregation
14	MFA	Measure of Functional Abstraction
15	CAM	Cohesion Among Methods
16	IC	Inheritance Coupling
17	CBM	Coupling Between Methods
18	AMC	Average Method Complexity
19	MAX_CC	Maximum cyclomatic complexity
20	AVG_CC	Mean cyclomatic complexity

The reason for using these datasets is driven by our goal to account for noise in the data, which is a threat specified by the donors of these datasets. Each dataset contains a number of instances corresponding to the classes in the release. Originally, each instance has 20 static code metrics listed in Table 3. Three scenarios were considered for selecting the metric suites. In the first scenario, we used CK+LOC portion of the metrics as the basis of our experiments. CK+LOC is used and validated in previous CPDP studies [2, 46] and Hall et al. [1] have addressed the usefulness of these metrics in comparison with static code metrics. In the second scenario, the full set of metrics were considered for our experiments and finally for the last scenario, we used a very simple and fast feature selection approach based on the rank of the features according to their information gain. The selection of the metrics in our original study was skipped as its primary focus was only on the instance selection problem and using a reduced set that is tried in other studies allowed us to demonstrate the feasibility of our approach as a proof of concept. While this paper includes the same feature set, it also involves the feature selection concept to some extent and detailed analysis are presented accordingly.

3.5. Performance Measures and Tools

Naïve Bayes (NB) is used as the base learner in all experiments. NB is a member of the probabilistic classifier family that are based on applying Bayes' theorem with strong (naïve) independence assumptions between the features [47]. The good performance of NB has been shown in many studies. Menzies et al. [3, 48] and Lessmann et al.[49] have demonstrated the effectiveness of NB with a set of data mining experiments performed on NASA MDP datasets. Lessmann et al. compared the most common classifiers on the NASA datasets and concluded that there is no significant difference between the performances of top 15 classifiers, one of which is NB [49] .

To assess the performance of the models, four indicators are used: Precision, Recall, F1 and G. These indicators are calculated by comparing the outcome of the prediction model and the actual label of the data instances. To that end, the confusion matrix is created using the following values:

TN: The number of **correct** predictions that instances are defect free.

FN: The number of **incorrect** predictions that instances are defect free.

TP: The number of **correct** predictions that instances are defective.

FP: The number of **incorrect** predictions that instances are defective.

Using confusion matrix, mentioned indicators are calculated as follows:

Precision: The proportion of the predicted positive cases that were correct is calculated using:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

Recall: Recall is the proportion of positive cases that were correctly identified. To calculate recall the following equation is used:

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

F1: To capture the trade-off between precision and recall, F1 (F-Measure) is calculated using the values of recall and precision. The most common version of this measure is the F1-score which is the harmonic mean of precision and recall. This measure is approximately the average of the two when they are close, and is more generally the square of the geometric mean divided by the arithmetic mean. We denote F1-measure by F1 in the following.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5}$$

G: While F1 is the harmonic mean of Recall and Precision, G (GMean) is the geometric mean of the two.

$$G = \sqrt{precision \times recall} \tag{6}$$

In this study, F1 and G are selected as the principal measures of reporting our results and performing comparisons in order to detect the best approach(es). F1 and G are also used as parts of the fitness function in GIS as discussed earlier. Finally, F1 is used in the context of the iterative infogain subsetting to select the best set of features according to their information gain.

All the experiments are conducted using **WEKA**[1] machine learning library version 3.6.13. The statistical tests are carried out using the **scipy.stats**[2] library version 0.16.1, **Python**[3] version 3.4.4 and **statistics** library from Python. The violin plots and CD Diagrams are generated using the **matplotlib**[4] library version 1.5.3 and **evaluation** package from **Orange**[5] library version 3.3.8 respectively. A replication package is available online for GIS [6].

4. Results

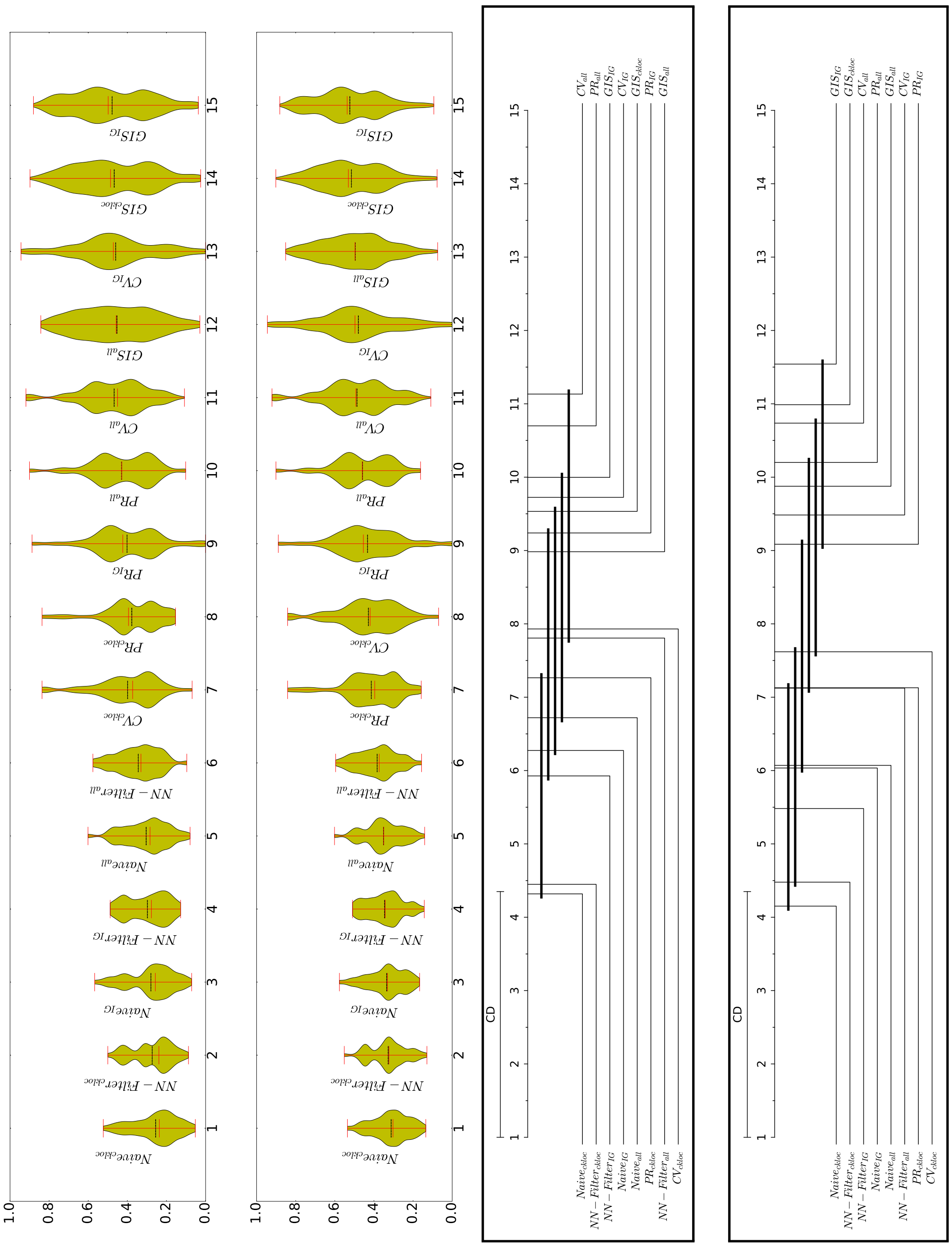
Tables 5, 6, 7 and 8 provide the median F1 and G values from the experiments performed for CPDP and WPDP benchmarks, respectively. In these tables, the reported results are without variation for (NN)-Filter and Naive CPDP methods as well as PR since there is no randomness involved in their settings. For other benchmarks, the experiments are repeated 30 times to account for the existing randomness in the design of their experiments. The results of within and cross project predictions are presented separately to evaluate the differences in both within and cross project cases and to answer the corresponding research questions properly. The results of GIS are duplicated in the cross and within project tables to make the comparisons easier. In both sets of tables, the last two rows present the median and mean values of all predictions.

These results are depicted through diagrams and plots in Figures 4 and 13. The rankings in the first figure plots are based on the median and critical difference scheme. The third figure provides per datasets results for GIS, CPDP and WPDP. These plots are described in the following.

---

[1] `http://www.cs.waikato.ac.nz/ml/weka/`
[2] `https://www.scipy.org/`
[3] `http://www.python.org`
[4] `http://matplotlib.org/`
[5] `http://orange.biolab.si/`
[6] `https://doi.org/10.5281/zenodo.804413`

Table 4: Violin Plots and CD Diagrams for F1 and G



(a) F1 (b) G (c) F1 (d) G

Table 5: F1: GIS vs Cross Project Benchmarks

	GIS(C)			NN-Filter(C)			Naive(C)		
file	**All**	**ckloc**	**IG**	**All**	**ckloc**	**IG**	**All**	**ckloc**	**IG**
ant-1.3	0.292	*0.326*	0.314	0.372	0.444	**0.488**	0.294	0.250	0.375
ant-1.4	***0.370***	0.355	0.361	0.219	0.185	0.274	0.190	0.129	0.197
ant-1.5	0.211	0.245	*0.251*	0.313	**0.444**	0.338	0.338	0.418	0.423
ant-1.6	0.442	0.465	***0.513***	0.410	0.426	0.450	0.408	0.420	0.444
ant-1.7	0.361	0.395	*0.416*	0.497	0.424	0.485	0.465	0.437	**0.504**
camel-1.0	*0.102*	0.078	0.089	0.188	0.238	0.128	**0.333**	**0.333**	0.194
camel-1.2	0.491	***0.525***	0.483	0.271	0.238	0.240	0.192	0.170	0.167
camel-1.4	0.302	0.303	***0.320***	0.281	0.239	0.269	0.204	0.201	0.246
camel-1.6	0.329	***0.342***	0.329	0.219	0.214	0.226	0.196	0.235	0.212
ivy-1.1	0.664	***0.703***	0.589	0.375	0.222	0.222	0.274	0.225	0.243
ivy-1.4	0.150	0.157	*0.173*	**0.318**	0.129	0.182	0.300	0.273	0.292
ivy-2.0	0.277	0.300	*0.338*	0.364	**0.434**	0.412	0.391	0.391	0.421
jedit-3.2	0.543	0.575	***0.587***	0.336	0.226	0.303	0.486	0.359	0.424
jedit-4.0	0.423	0.451	*0.463*	0.422	0.302	0.368	0.468	0.468	**0.500**
jedit-4.1	0.486	0.486	*0.520*	0.493	0.414	0.403	**0.601**	0.523	0.567
jedit-4.2	0.300	0.259	*0.342*	0.443	0.378	0.420	0.460	0.473	**0.481**
jedit-4.3	0.043	0.034	*0.047*	0.096	**0.164**	0.152	0.079	0.119	0.108
log4j-1.0	0.447	***0.523***	0.442	0.519	0.391	0.348	0.256	0.162	0.111
log4j-1.1	0.575	***0.619***	0.579	0.576	0.500	0.462	0.233	0.053	0.150
log4j-1.2	0.730	0.668	***0.781***	0.217	0.138	0.165	0.119	0.071	0.071
lucene-2.0	0.633	***0.640***	0.609	0.446	0.324	0.383	0.175	0.175	0.198
lucene-2.2	0.643	***0.680***	0.612	0.282	0.226	0.235	0.185	0.127	0.127
lucene-2.4	0.691	***0.714***	0.668	0.358	0.217	0.252	0.280	0.211	0.194
poi-1.5	0.681	0.706	***0.741***	0.314	0.210	0.210	0.284	0.200	0.222
poi-2.0	0.215	0.216	0.219	**0.267**	0.197	0.230	0.234	0.215	0.257
poi-2.5	0.768	0.761	***0.796***	0.233	0.165	0.179	0.262	0.176	0.246
poi-3.0	0.766	***0.803***	0.786	0.263	0.185	0.196	0.269	0.194	0.287
synapse-1.0	0.196	0.220	*0.223*	**0.421**	0.311	0.410	0.333	0.276	0.320
synapse-1.1	0.415	*0.457*	0.455	**0.463**	0.311	0.442	0.370	0.237	0.240
synapse-1.2	0.520	0.527	0.537	**0.560**	0.310	0.431	0.431	0.262	0.273
velocity-1.4	0.564	0.642	***0.724***	0.188	0.088	0.132	0.120	0.099	0.133
velocity-1.5	0.628	0.583	***0.712***	0.228	0.116	0.185	0.164	0.116	0.198
velocity-1.6	0.506	0.521	***0.558***	0.291	0.205	0.317	0.250	0.237	0.283
xalan-2.4	0.304	0.287	*0.311*	0.390	0.317	0.344	0.367	0.327	**0.400**
xalan-2.5	0.569	***0.583***	0.577	0.373	0.301	0.301	0.395	0.281	0.294
xalan-2.6	0.514	0.567	***0.589***	0.511	0.404	0.413	0.490	0.375	0.382
xalan-2.7	0.798	***0.831***	0.763	0.402	0.248	0.251	0.416	0.255	0.261
xerces-1.2	0.234	***0.256***	0.239	0.240	0.171	0.200	0.244	0.200	0.240
xerces-1.3	***0.379***	0.329	0.327	0.331	0.291	0.288	0.331	0.327	0.295
xerces-1.4	0.646	0.638	***0.710***	0.310	0.189	0.198	0.250	0.171	0.184
xerces-init	0.408	0.433	***0.516***	0.318	0.258	0.277	0.318	0.295	0.303
Median	0.457	0.486	0.498	0.331	0.239	0.277	0.284	0.237	0.257
Mean	0.453	0.467	0.478	0.344	0.273	0.298	0.304	0.255	0.280

To measure the performance difference across the benchmarks, two different approaches were considered. First, the performance of GIS in comparison with other benchmarks was assessed through Wilcoxon signed rank tests. Tables 9 and 10 summarize the results of the pairwise statistical tests based on F1 and G values respectively for all 30 runs. The first column of each entry in these tables is the $p - value$ obtained from the tests and the second column is the Cohen's d value associated with the performance obtained from the two treatments subject to comparison. The following equation is used to

Table 6: F1: GIS vs Within Project Benchmarks

	GIS (C)			CV (W)			PR (W)		
file	**All**	**ckloc**	**IG**	**All**	**ckloc**	**IG**	**All**	**ckloc**	**IG**
ant-1.3	0.292	0.326	0.314	0.427	0.303	**0.441**	0.427	0.303	**0.441**
ant-1.4	0.370	0.355	0.361	0.400	0.394	**0.444**	0.308	0.154	0.278
ant-1.5	0.211	0.245	0.251	0.370	0.448	**0.507**	0.429	0.430	0.500
ant-1.6	0.442	0.465	0.513	0.576	0.431	0.586	**0.601**	0.514	0.477
ant-1.7	0.361	0.395	0.416	**0.556**	0.497	0.498	0.531	0.438	0.518
camel-1.0	0.102	0.078	0.089	**0.300**	0.286	0.118	**0.300**	0.286	0.118
camel-1.2	0.491	**0.525**	0.483	0.322	0.288	0.205	0.208	0.178	0.053
camel-1.4	0.302	0.303	**0.320**	0.265	0.245	0.264	0.300	0.304	0.288
camel-1.6	0.329	**0.342**	0.329	0.312	0.261	0.204	0.306	0.287	0.259
ivy-1.1	0.664	**0.703**	0.589	0.574	0.449	0.538	0.574	0.449	0.538
ivy-1.4	0.150	0.157	0.173	0.176	0.080	0.000	0.267	0.250	**0.278**
ivy-2.0	0.277	0.300	0.338	0.389	**0.425**	**0.425**	0.375	0.380	0.424
jedit-3.2	0.543	0.575	**0.587**	0.572	0.467	0.462	0.572	0.467	0.462
jedit-4.0	0.423	0.451	0.463	0.421	0.294	0.237	**0.517**	0.394	0.481
jedit-4.1	0.486	0.486	0.520	0.500	0.361	0.398	**0.526**	0.400	0.323
jedit-4.2	0.300	0.259	0.342	0.432	0.320	0.400	**0.475**	0.405	0.465
jedit-4.3	0.043	0.034	0.047	0.211	0.214	0.077	0.102	0.164	**0.233**
log4j-1.0	0.447	0.523	0.442	**0.632**	0.607	0.584	**0.632**	0.607	0.584
log4j-1.1	0.575	0.619	0.579	**0.725**	0.687	0.697	0.708	0.698	0.667
log4j-1.2	0.730	0.668	**0.781**	0.657	0.578	0.686	0.453	0.417	0.474
lucene-2.0	0.633	**0.640**	0.609	0.553	0.507	0.556	0.553	0.507	0.556
lucene-2.2	0.643	**0.680**	0.612	0.487	0.423	0.451	0.500	0.452	0.426
lucene-2.4	0.691	**0.714**	0.668	0.529	0.466	0.526	0.525	0.435	0.525
poi-1.5	0.681	0.706	**0.741**	0.454	0.409	0.592	0.454	0.409	0.592
poi-2.0	0.215	0.216	0.219	0.207	0.218	0.162	**0.288**	0.269	**0.288**
poi-2.5	0.768	0.761	0.796	0.578	0.281	**0.812**	0.256	0.208	0.238
poi-3.0	0.766	**0.803**	0.786	0.477	0.369	0.688	0.294	0.264	0.338
synapse-1.0	0.196	0.220	0.223	0.385	0.296	**0.432**	0.385	0.296	**0.432**
synapse-1.1	0.415	0.457	0.455	**0.527**	0.433	0.461	0.500	0.427	0.469
synapse-1.2	0.520	0.527	0.537	**0.577**	0.505	0.530	0.510	0.397	0.489
velocity-1.4	0.564	0.642	0.724	**0.892**	0.831	0.880	**0.892**	0.831	0.880
velocity-1.5	0.628	0.583	0.712	0.433	0.311	0.494	0.752	0.756	**0.765**
velocity-1.6	0.506	0.521	**0.558**	0.360	0.305	0.410	0.526	0.505	0.530
xalan-2.4	0.304	0.287	0.311	**0.363**	0.299	0.354	**0.363**	0.299	0.354
xalan-2.5	0.569	**0.583**	0.577	0.377	0.302	0.555	0.306	0.297	0.301
xalan-2.6	0.514	0.567	0.589	**0.598**	0.535	0.575	0.428	0.407	0.407
xalan-2.7	0.798	0.831	0.763	0.913	0.820	**0.930**	0.349	0.260	0.285
xerces-1.2	0.234	**0.256**	0.239	0.231	0.175	0.162	0.231	0.175	0.162
xerces-1.3	0.379	0.329	0.327	0.372	0.274	**0.466**	0.291	0.265	0.247
xerces-1.4	0.646	0.638	0.710	**0.718**	0.645	0.707	0.254	0.189	0.000
xerces-init	0.408	0.433	**0.516**	0.333	0.327	0.330	0.349	0.312	0.362
Median	0.457	0.486	0.498	0.450	0.373	0.472	0.429	0.394	0.424
Mean	0.453	0.467	0.478	0.468	0.399	0.460	0.430	0.377	0.402

calculate Cohen's d:

$$d = \frac{X_{gis} - X_b}{std_p} \tag{7}$$

Here, X_b and X_{gis} are the means of the benchmark and GIS respectively. Hence, a positive Cohen's d means that GIS yields better results than the compared counterpart. std_p represents the pooled standard deviation which

Table 7: G: GIS vs Cross Project Benchmarks

	GIS(C)			NN-Filter(C)			Naive(C)		
file	**All**	**ckloc**	**IG**	**All**	**ckloc**	**IG**	**All**	**ckloc**	**IG**
ant-1.3	0.381	*0.434*	0.418	0.373	0.447	**0.488**	0.299	0.258	0.387
ant-1.4	***0.446***	0.397	0.410	0.220	0.190	0.275	0.198	0.135	0.207
ant-1.5	0.314	0.353	*0.359*	0.322	**0.447**	0.343	0.340	0.418	0.425
ant-1.6	0.507	0.528	***0.558***	0.417	0.447	0.461	0.422	0.438	0.463
ant-1.7	0.450	0.486	*0.501*	0.502	0.449	0.504	0.477	0.454	**0.523**
camel-1.0	*0.208*	0.193	0.201	0.233	0.258	0.143	**0.347**	**0.347**	0.196
camel-1.2	0.520	***0.580***	0.512	0.299	0.299	0.292	0.256	0.257	0.242
camel-1.4	0.393	***0.416***	0.388	0.285	0.266	0.282	0.212	0.226	0.266
camel-1.6	0.402	***0.445***	0.388	0.226	0.246	0.249	0.207	0.267	0.234
ivy-1.1	0.664	***0.705***	0.604	0.458	0.336	0.336	0.398	0.356	0.342
ivy-1.4	0.247	0.270	*0.271*	**0.331**	0.129	0.182	0.306	0.283	0.309
ivy-2.0	0.366	0.391	*0.419*	0.371	**0.434**	0.419	0.395	0.395	0.426
jedit-3.2	0.563	***0.612***	0.603	0.374	0.274	0.352	0.502	0.393	0.455
jedit-4.0	0.467	0.502	***0.515***	0.428	0.332	0.388	0.469	0.478	0.511
jedit-4.1	0.520	0.529	*0.561*	0.501	0.444	0.427	**0.602**	0.536	0.576
jedit-4.2	0.389	0.364	*0.435*	0.453	0.379	0.420	**0.484**	0.477	**0.484**
jedit-4.3	0.114	0.098	*0.129*	0.156	**0.213**	0.203	0.141	0.176	0.166
log4j-1.0	0.494	***0.565***	0.517	0.537	0.446	0.396	0.383	0.297	0.243
log4j-1.1	0.581	***0.635***	0.615	0.596	0.552	0.509	0.336	0.164	0.285
log4j-1.2	0.746	0.697	***0.790***	0.349	0.272	0.300	0.252	0.192	0.192
lucene-2.0	0.638	***0.654***	0.610	0.517	0.422	0.471	0.272	0.272	0.331
lucene-2.2	0.643	***0.681***	0.614	0.363	0.323	0.327	0.295	0.231	0.231
lucene-2.4	0.693	***0.718***	0.669	0.439	0.338	0.356	0.377	0.344	0.315
poi-1.5	0.681	0.709	***0.748***	0.408	0.312	0.312	0.382	0.309	0.331
poi-2.0	0.287	0.318	***0.327***	0.267	0.201	0.235	0.234	0.217	0.258
poi-2.5	0.769	0.762	***0.803***	0.325	0.267	0.285	0.350	0.265	0.341
poi-3.0	0.767	***0.809***	0.794	0.361	0.301	0.308	0.365	0.300	0.390
synapse-1.0	0.292	0.337	*0.343*	**0.469**	0.325	0.417	0.334	0.277	0.333
synapse-1.1	0.461	***0.521***	0.514	0.466	0.330	0.458	0.388	0.290	0.300
synapse-1.2	0.554	0.574	***0.577***	0.566	0.354	0.455	0.455	0.329	0.330
velocity-1.4	0.575	0.645	***0.725***	0.275	0.160	0.203	0.189	0.176	0.214
velocity-1.5	0.635	0.602	***0.712***	0.319	0.209	0.281	0.265	0.209	0.300
velocity-1.6	0.511	0.538	***0.569***	0.340	0.322	0.378	0.320	0.322	0.346
xalan-2.4	0.392	0.385	***0.402***	0.397	0.322	0.347	0.383	0.327	0.400
xalan-2.5	0.575	***0.590***	0.583	0.399	0.360	0.360	0.414	0.331	0.344
xalan-2.6	0.522	0.580	***0.596***	0.540	0.478	0.486	0.509	0.440	0.445
xalan-2.7	0.813	***0.842***	0.785	0.502	0.376	0.379	0.513	0.382	0.388
xerces-1.2	0.249	0.278	***0.287***	0.242	0.183	0.201	0.247	0.209	0.242
xerces-1.3	***0.428***	0.356	0.400	0.334	0.310	0.290	0.334	0.338	0.298
xerces-1.4	0.665	0.652	***0.716***	0.418	0.308	0.310	0.371	0.303	0.301
xerces-init	0.409	0.436	***0.519***	0.354	0.342	0.326	0.354	0.376	0.364
Median	0.497	0.531	0.537	0.373	0.323	0.343	0.350	0.303	0.331
Mean	0.496	0.515	0.523	0.384	0.327	0.345	0.351	0.312	0.335

can be calculated as follows:

$$std_p = \sqrt{\frac{(n_{gis} - 1) * (s_{gis})^2 + (n_b - 1) * (s_b)^2}{n_{gis} + n_b - 2}} \tag{8}$$

Where s_{gis}, n_{gis}, s_b and n_b are the standard deviation of GIS measurements, the number of subjects in the GIS group, standard deviation of benchmark group and number of subjects in the benchmark group, respectively. Cohen's d is a way of representing the standardized difference between

Table 8: G: GIS vs Within Project Benchmarks

	GIS (C)			CV (W)			PR (W)		
file	**All**	**ckloc**	**IG**	**All**	**ckloc**	**IG**	**All**	**ckloc**	**IG**
ant-1.3	0.381	0.434	0.418	0.429	0.310	**0.442**	0.429	0.310	**0.442**
ant-1.4	0.446	0.397	0.410	0.451	0.454	**0.500**	0.308	0.158	0.280
ant-1.5	0.314	0.353	0.359	0.411	0.448	**0.511**	0.457	0.438	0.506
ant-1.6	0.507	0.528	0.558	0.577	0.448	0.590	**0.604**	0.529	0.489
ant-1.7	0.450	0.486	0.501	**0.556**	0.508	0.508	0.532	0.463	0.528
camel-1.0	0.208	0.193	0.201	**0.320**	0.296	0.139	**0.320**	0.296	0.139
camel-1.2	0.520	**0.580**	0.512	0.349	0.336	0.257	0.280	0.269	0.118
camel-1.4	0.393	**0.416**	0.388	0.269	0.264	0.288	0.301	0.312	0.303
camel-1.6	0.402	**0.445**	0.388	0.326	0.285	0.237	0.311	0.306	0.277
ivy-1.1	0.664	**0.705**	0.604	0.593	0.494	0.570	0.593	0.494	0.570
ivy-1.4	0.247	0.270	0.271	0.177	0.083	0.000	0.325	0.301	**0.334**
ivy-2.0	0.366	0.391	0.419	0.393	**0.425**	**0.425**	0.380	0.380	0.424
jedit-3.2	0.563	**0.612**	0.603	0.580	0.479	0.475	0.580	0.479	0.475
jedit-4.0	0.467	0.502	0.515	0.432	0.333	0.299	**0.517**	0.396	0.485
jedit-4.1	0.520	0.529	**0.561**	0.514	0.411	0.453	0.532	0.431	0.403
jedit-4.2	0.389	0.364	0.435	0.433	0.333	0.408	**0.483**	0.409	0.468
jedit-4.3	0.114	0.098	0.129	0.232	0.219	0.078	0.162	0.213	**0.267**
log4j-1.0	0.494	0.565	0.517	**0.644**	0.622	0.597	**0.644**	0.622	0.597
log4j-1.1	0.581	0.635	0.615	**0.727**	0.690	0.702	0.715	0.709	0.677
log4j-1.2	0.746	0.697	**0.790**	0.694	0.630	0.718	0.535	0.509	0.554
lucene-2.0	0.638	**0.654**	0.610	0.572	0.544	0.583	0.572	0.544	0.583
lucene-2.2	0.643	**0.681**	0.614	0.517	0.481	0.490	0.536	0.506	0.471
lucene-2.4	0.693	**0.718**	0.669	0.570	0.524	0.561	0.560	0.502	0.560
poi-1.5	0.681	0.709	**0.748**	0.505	0.474	0.608	0.505	0.474	0.608
poi-2.0	0.287	0.318	0.327	0.208	0.232	0.186	0.306	0.281	**0.331**
poi-2.5	0.769	0.762	0.803	0.608	0.357	**0.812**	0.345	0.301	0.328
poi-3.0	0.767	**0.809**	0.794	0.529	0.449	0.699	0.388	0.364	0.427
synapse-1.0	0.292	0.337	0.343	0.420	0.303	**0.436**	0.420	0.303	**0.436**
synapse-1.1	0.461	0.521	0.514	**0.528**	0.446	0.474	0.506	0.455	0.482
synapse-1.2	0.554	0.574	0.577	**0.580**	0.519	0.543	0.516	0.426	0.514
velocity-1.4	0.575	0.645	0.725	**0.893**	0.835	0.881	**0.893**	0.835	0.881
velocity-1.5	0.635	0.602	0.712	0.490	0.390	0.538	0.761	0.765	**0.775**
velocity-1.6	0.511	0.538	0.569	0.394	0.349	0.435	0.557	0.520	**0.593**
xalan-2.4	0.392	0.385	**0.402**	0.363	0.304	0.356	0.363	0.304	0.356
xalan-2.5	0.575	**0.590**	0.583	0.409	0.353	0.556	0.352	0.364	0.360
xalan-2.6	0.522	0.580	0.596	**0.625**	0.577	0.613	0.479	0.476	0.486
xalan-2.7	0.813	0.842	0.785	0.917	0.833	**0.932**	0.460	0.386	0.407
xerces-1.2	0.249	0.278	**0.287**	0.235	0.181	0.181	0.235	0.181	0.181
xerces-1.3	0.428	0.356	0.400	0.375	0.278	**0.466**	0.295	0.272	0.273
xerces-1.4	0.665	0.652	0.716	**0.738**	0.674	0.733	0.374	0.308	0.000
xerces-init	0.409	0.436	**0.519**	0.387	0.398	0.388	0.383	0.392	0.383
Median	0.497	0.531	0.537	0.492	0.420	0.497	0.460	0.396	0.454
Mean	0.496	0.515	0.523	0.487	0.428	0.480	0.459	0.414	0.433

two groups. It is usually used alongside a statistical test (in this case, the Wilcoxon tests) as a measure of magnitude of differences. Sawilowsky [50] describes the magnitudes of the effect size in six categories by extending the original three [51]. The six categories are: very small (0.01), small (0.2) medium (0.5), large (0.8), very large (1.2) and huge (2.0).

Please note that the measurements are copied multiple times in order to have comparable groups for comparisons in case of NN-Filter and Naive CPDP as no randomness in involved in their settings. Additionally, the results for the first release of each project in PR benchmark is copied from the same counterpart in CV. In that case, one might see a slight variation in the results even though there is no actual randomness involved, something that we have accounted for, in our analysis.

The overall performance of all presented approaches and their possible differences were investigated through a second set of tests for comparison of multiple groups. To that end, we first perform Friedman non-parametric test [52] to detect any significant difference across the compared groups. The Friedman test works on average ranks and tests for significant differences within the compared groups. The Friedman test can be done via the following equations [53]:

$$\chi_F^2 = \frac{12 \times N}{k(k+1)} \left(\sum_j {R_j}^2 - \frac{k(k+1)^2}{4} \right) \tag{9}$$

$$F_F = \frac{(N-1) \times \chi_F^2}{N \times (k-1) - \chi_F^2} \tag{10}$$

In these equations, N and k are the number of instances (41 datasets in our experiments) and the number of compared groups (15 groups, three for each benchmark) respectively. F_F which uses Friedman's chi-square statistic is distributed according to the F distribution with $(k-1)$ and $(k-1)\times(N-1)$ degrees of freedom. Despite detecting the existence of significant differences, the Friedman test is not able to locate their positions. If the null hypothesis, i.e. all groups perform similarly, is rejected, the search for the location of possible differences continues with extra tests. Since we compare all of the groups against each other, Nemenyi's post-hoc test [54] is used in case of observing significant differences. This test is different from Bonferroni-Dunn test where a control group is compared against other groups [53]. With Nemenyi's test, a critical difference is calculated from the average ranks as well as the number of datasets that are utilized during the experiments. The following equation is used for calculating Nemenyi's critical difference values in different levels of significance[53]:

$$CD = q_{\alpha,k} \sqrt{\frac{k(k+1)}{6 \times N}} \tag{11}$$

Acquired $CD = 3.3496$ depends on $q_\alpha = 3.39123$ which in turn is dependent to the number of compared groups ($k = 15$) as well as the significance level used for the comparisons ($\alpha = 0.05$). Each two approaches are significantly different whenever their average ranks differ by at least one CD. The Friedman test in conjunction with Nemenyi's test rank the approaches with the highest rank belonging to the best performing approach to the lowest based on their average ranks. The results of these tests are presented through CD diagrams in Figure 4 for F1 and G.

Beside these tests, another set of statistical tests were used to detect different levels of significance among individual datasets. We used Kruskal Wallis H (KW-H) test to detect such differences. Similar to the Friedman test, one should note the limited power of such tests from two aspects. First, KW-H is a non parametric test and has less power in comparison with its parametric counterpart, i.e. One way ANOVA. Secondly, KW-H only shows whether a difference could be observed at a specific confidence level and is not able to detect the position of such differences. To identify those positions, extra tests such as Nemenyi's post-hoc test or Bonferroni-Dunn test are required depending on how the comparisons are done. We skipped to perform such tests in this case for two reasons. First, performing and analysing such test for individual datasets makes the analysis very complicated. Secondly, the structure of the reported results through tables grouped by the benchmarks makes it very difficult to present any form of visualization for such cases. Instead, if we detect a significant difference, we report the group with the highest median as the best treatment for that particular dataset. Further, as pointed out earlier, we copied the measurements from 10 fold WP cross validation for the first releases of each project for PR benchmark. Hence, multiple treatments are selected as best in some cases since they are identical. Per dataset performances are illustrated in Figure 13 separated into GIS, CPDP and WPDP categories.

The results of the experiments are also visualized in violin plots [55]. Even though violin plots are in some sense similar to box plots, they are more informative. A box plot only shows the summary statistics such as mean/median and inter-quartile ranges while the violin plot shows the full distribution of the data. Note that the thin continuous line in the plots is the median and the thick dashed line represents the mean value of the results.

Based on the results achieved, the research questions are answered as follows.

Table 9: Wilcoxon signed rank test results and effect sizes for the pairwise comparison between GIS and other benchmarks in terms of F1. Positive effect sizes point to an effect size in favor of GIS.

	$\mathbf{GIS_{all}}$		$\mathbf{GIS_{ckloc}}$		$\mathbf{GIS_{IG}}$	
	p-value	**d**	**p-value**	**d**	**p-value**	**d**
$\mathbf{CV_{all}}$	0.000	-0.102	0.202	-0.004	0.451	0.071
$\mathbf{CV_{ckloc}}$	0.000	0.319	0.000	0.413	0.000	0.472
$\mathbf{CV_{IG}}$	0.375	-0.046	0.073	0.055	0.000	0.136
$\mathbf{NN\text{-}Filter_{all}}$	0.000	0.522	0.000	0.590	0.000	0.621
$\mathbf{NN\text{-}Filter_{ckloc}}$	0.000	0.760	0.000	0.817	0.000	0.845
$\mathbf{NN\text{-}Filter_{IG}}$	0.000	0.668	0.000	0.726	0.000	0.762
$\mathbf{Naive_{all}}$	0.000	0.647	0.000	0.700	0.000	0.743
$\mathbf{Naive_{ckloc}}$	0.000	0.776	0.000	0.818	0.000	0.865
$\mathbf{Naive_{IG}}$	0.000	0.692	0.000	0.738	0.000	0.789
$\mathbf{PR_{all}}$	0.527	0.117	0.005	0.189	0.000	0.249
$\mathbf{PR_{ckloc}}$	0.000	0.365	0.000	0.436	0.000	0.505
$\mathbf{PR_{IG}}$	0.000	0.230	0.000	0.297	0.000	0.355

Table 10: Wilcoxon signed rank test results and effect sizes for the pairwise comparison between GIS and other benchmarks in terms of G. Positive effect sizes point to an effect size in favor of GIS.

	$\mathbf{GIS_{all}}$		$\mathbf{GIS_{ckloc}}$		$\mathbf{GIS_{IG}}$	
	p-value	**d**	**p-value**	**d**	**p-value**	**d**
$\mathbf{CV_{all}}$	0.001	0.075	0.000	0.243	0.000	0.325
$\mathbf{CV_{ckloc}}$	0.000	0.503	0.000	0.665	0.000	0.739
$\mathbf{CV_{IG}}$	0.000	0.131	0.000	0.288	0.000	0.387
$\mathbf{NN\text{-}Filter_{all}}$	0.000	0.729	0.000	0.874	0.000	0.899
$\mathbf{NN\text{-}Filter_{ckloc}}$	0.000	0.969	0.000	1.095	0.000	1.114
$\mathbf{NN\text{-}Filter_{IG}}$	0.000	0.894	0.000	1.024	0.000	1.056
$\mathbf{Naive_{all}}$	0.000	0.867	0.000	0.980	0.000	1.027
$\mathbf{Naive_{ckloc}}$	0.000	0.975	0.000	1.070	0.000	1.119
$\mathbf{Naive_{IG}}$	0.000	0.884	0.000	0.989	0.000	1.050
$\mathbf{PR_{all}}$	0.000	0.232	0.000	0.364	0.000	0.438
$\mathbf{PR_{ckloc}}$	0.000	0.502	0.000	0.640	0.000	0.732
$\mathbf{PR_{IG}}$	0.000	0.331	0.000	0.448	0.000	0.511

Table 11: GIS vs GIS

	$\mathbf{GIS_{all}}$ vs. $\mathbf{GIS_{ckloc}}$		$\mathbf{GIS_{all}}$ vs. $\mathbf{GIS_{IG}}$		$\mathbf{GIS_{ckloc}}$ vs. $\mathbf{GIS_{IG}}$	
	p-value	**d**	**p-value**	**d**	**p-value**	**d**
F1	0.000	-0.280	0.000	-0.413	0.000	-0.168
G	0.000	-0.366	0.000	-0.471	0.000	-0.134

4.1. RQ1: How is the performance of GIS compared with benchmark cross project defect prediction approaches?

Table 5 presents the results of GIS and cross project benchmarks. Category wise, GIS outperforms CPDP benchmarks in 26 cases by achieving the highest median values while the KW-H tests show the existence of a significant difference. (NN)-Filter has a better performance in nine cases and the

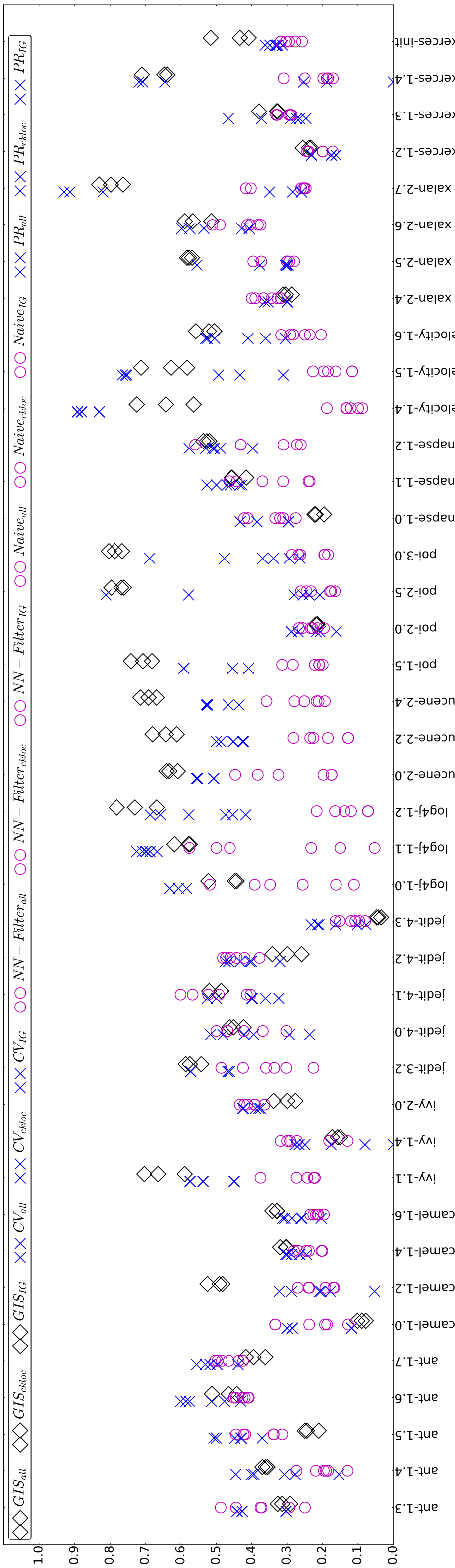
Table 12: Selected features and their order for individual projects for GIS, Naive CPDP and NN-Filter

For Project	Selected Features
ant	LOC, RFC, LCOM3, AMC, WMC, CAM, NPM, DAM, MAX_CC, LCOM
camel	LOC, RFC, LCOM3, CAM, WMC, MFA
ivy	LOC, RFC, LCOM3, WMC, CAM, AMC
jedit	LOC, RFC, AMC
log4j	LOC, RFC, LCOM3, WMC, AMC, CAM, NPM, LCOM
lucene	LOC, RFC, LCOM3, WMC, CAM, AMC
poi	LOC, RFC, AMC, CAM, LCOM3, WMC, NPM, MAX_CC, LCOM
synapse	LOC, RFC, WMC, CAM, LCOM3, AMC
velocity	LOC, RFC, LCOM3, AMC, WMC, CAM, NPM, MAX_CC, LCOM
xalan	LOC, RFC, WMC, CAM, NPM, LCOM3, LCOM
xerces	LOC, RFC, WMC, LCOM3, AMC, NPM, CAM, LCOM

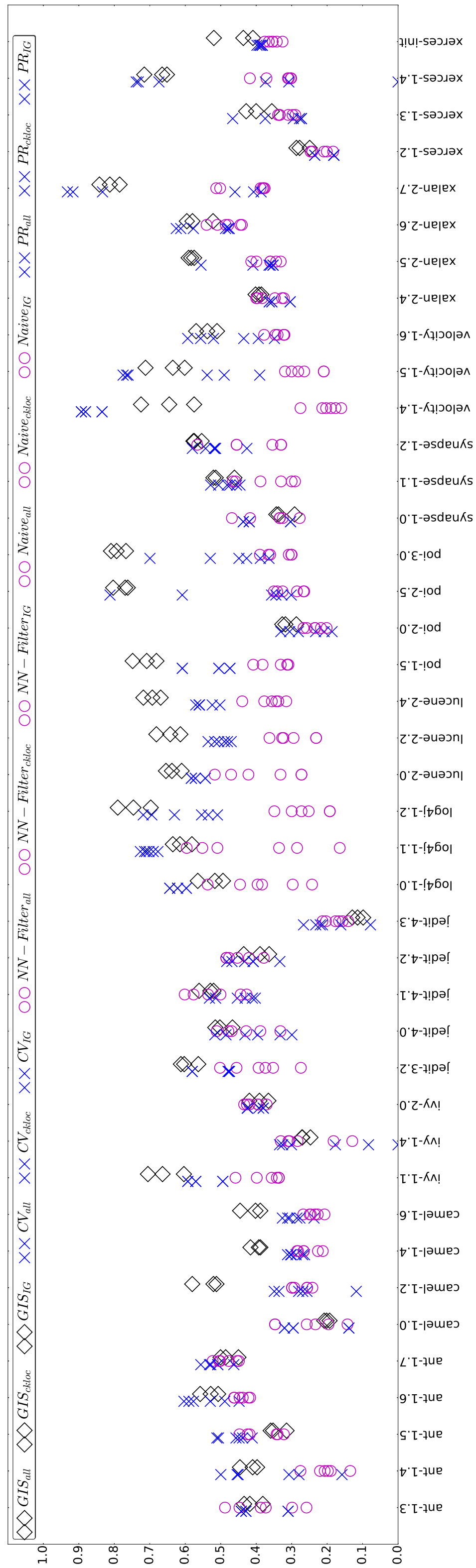
five remaining cases are in favor of Naive CPDP. With G, the performance of GIS is even better. The number of test sets that have better predictions are increased to 31 out of 41 for G. (NN)-Filter has six and naive CPDP has four better predictions. The overall mean and median values from GIS are higher than that of both benchmark cross project methods for F1 and G values with respect to all the metric sets.

The violin plots of the measurements for F1 and G values in Figure 4 provide more insights into the results. GIS variants have higher mean, median and max values compared with the CP benchmark methods. More specifically, they provide the first, second and fourth highest median F1 and first, second and third highest G values while the best CPDP benchmark in terms of median F1 and G has the tenth rank. Of course one should note also the weak performances on a couple of datasets and the drop in the minimum value with GIS as well as its wider range of the predictions. From the results in Tables 7 and 5, we can see that GIS has difficulties in predicting datasets like JEdit-4.3 (median F1 around 0.04) and Camel-1.0 (median F1 around 0.09). At the same time, the good performances of GIS on datasets from Xalan, Velocity, Synapse and Poi projects to name some, cause a dramatic increase in the max value. Nevertheless, the concentration of the prediction results with GIS is promising, i.e., around half of all predictions are over the maximum values received by the CPDP benchmark methods. Another depiction of the performance difference between GIS and CPDP benchmarks can be seen in Figure 13 with respect to individual datasets. For the majority parts, the location of GIS points are higher than those from CPDP benchmarks while the performance differences in many cases are substantial (e.g for Lucene, Poi, Xalan, Velocity).

Table 13: Median F1 and G values of the benchmarks for individual datasets



(a) F1



(b) G

Results of the pairwise statistical tests and the calculated effect sizes show that GIS is significantly better than both benchmark CPDP approaches and the effect sizes confirm this conclusion. GIS_{all} which achieves the lowest performance in the GIS group, outperforms (NN)-Filter (all, ckloc, IG) in terms of F1 ($p - value \ll 0.001$, Cohen's $d = \{0.522, 0.760, 0.668\}$) and G ($p - value \ll 0.001$, Cohen's $d = \{0.729, 0.969, 0.894\}$). It also outperforms naive CPDP (all, ckloc, IG) in terms of F1 ($p - value \ll 0.001$, Cohen's $d = \{0.647, 0.776, 0.692\}$) and G ($p - value \ll 0.001$, Cohen's $d = \{0.0.867, 0.975, 0.884\}$). These performance improvements are more visible with G and the effect sizes are larger. With careful selection of the features, GIS could achieve even better results. Iterative IG in GIS_{IG} leads to better predictions in comparison with (NN)-Filter (best case) by providing the effect sizes of 0.621 and 0.899 in terms of F1 and G. Similarly, It outperforms Naive CPDP (best case) with the effect sizes of 0.743 and 1.027 with F1 and G.

The Friedman and Nemenyi tests for F1 and G confirm most of our findings from the pairwise Wilcoxon tests. One change in this case is that in terms of F1, $\text{NN-Filter}_{\text{all}}$ despite its lower average rank is present among the top ranking group of benchmarks. This situation does not occur with G and the top group contains only GIS and other WPDP benchmarks. Having said that, GIS benchmarks provide the highest absolute and per metric set ranks as well as highest rank sums (F1=(3+4+7) and G=(1+2+5)), outperforming both NN-Filter (F1=(9+13+14) and G=(10+13+14)) and Naive CPDP (F1=(11+12+15) and G=(11+12+15)).

Even though not presented in the tables, we should note that GIS is more focused on recall and has a lower precision while the benchmark approaches focus more on precision and have lower recall values. Our fitness function is defined in a way that treats the recall and precision equally, but previous studies have shown that the (NN)-Filter (on which GIS is optimized) focuses on recall more than precision [4]. A fitness function with more focus on precision could optimize the results for achieving values with higher precisions. Of course, this might come with a decrease in the recall as there usually is a trade-off between the two, but careful fitness function selection is one of the key areas to pursue further. This recall based nature could probably be linked to the choice of metrics as well (e.g. MODEP in [40] with CK+LOC metrics is heavily recall based and He at al.[26] asserted that feature selection in these datasets could be related to some degree of loss in precision).

4.2. RQ2: How is the performance of GIS compared with the within project defect prediction approach?

In terms of F1, the GIS category is better than both benchmark WPDP approaches in 15 cases while CV and PR provide better predictions in 18 and eight cases. With G, GIS is better in 16 cases whereas CV and PR are better in 17 and eight cases respectively.

The mean and median values from GIS are higher except in one case (CV_{all}) when compared based on the feature sets used. The worst case of GIS outperforms the best cases of PR in terms of both F1 and G.

The pairwise Wilcoxon tests provide a better insight into the results. GIS and WPDP benchmarks do not have a significant difference in five cases. The obtained $p-values = \{0.375, 0.527, 0.202, 0.073, 0.451\}$ as well as small effect sizes provide the evidence for such insignificant differences. The performance of one GIS variant, i.e. GIS_{all} is lower than one WPDP case, i.e. CV_{all} with a very small to small effect size (0.102). In all other cases, GIS outpeforms WPDP wherever a significant $p - value$ is observed with effect sizes ranging from 0.136 to 0.505.

With G, GIS is significantly better than all benchmark WPDP cases, but in some cases the effect sizes are very small. GIS has a tiny difference with CV_{all} (which provides the best WPDP performances considering both median and the range of the values-stability) based on the obtained effect size (0.075). Despite that, higher and significant effect sizes could be observed as well in case of CV_{ckloc} and PR_{ckloc} with effect sizes 0.739 and 0.732 (medium to large) respectively.

According to Friedman and Nemenyi tests for F1, GIS_{IG}, the third rank among all the benchmarks, achives the highest average rank among all CPDP approaches. The two other GIS variants, i.e. GIS_{ckloc} and GIS_{all} have the 5^{th} and 7^{th} ranks based on their average ranks. These GIS variants are accompanied by five WPDP benchmarks and one CPDP benchmark, i.e. NN-Filter$_{all}$ in the top ranking group of approaches for which no significant difference could be observed at $\alpha = 0.05$ with Nemenyi's test. This behaviour for the most part is in accordance with the Wilcoxon tests and further confirms our findings (except the presence of NN-Filter$_{all}$ in the top ranking group). With G, the top two ranks belong to GIS_{IG} and GIS_{ckloc} followed by four WPDP benchmark and the remaining GIS variant in the top performing group based on the Nemenyi's test. The rank sum for GIS in terms of F1=(3+5+7) is lower than CV with F1=(1+4+8) while GIS achieves higher rank sum with

G=(1+2+5) compared with CV which achieves G=(3+6+8). GIS outperforms PR in terms of rank sums with both F1=(2+6+10) and G=(4+7+9).

The shape of the violin plots also support our claims as illustrated in Figure 4. With F1, the GIS_{IG} and GIS_{ckloc} achieve the first and second highest highest median values and GIS_{all} gets the fourth spot after CV_{all}. With regard to G, the GIS variants manage to get the top three spots by outperforming all other benchmarks. GIS and WPDP benchmarks are overall less stable, but on the bright side, this instability is generally toward increasing the prediction performance. While CPDP benchmarks provide more stable predictions, their performance is significantly lower than GIS and WPDP.

Finally, the competitive behaviour of GIS variants compared with WPDP can be seen in Figure 13 as they usually perform better or as good as WPDP. Despite that, the lower performance with datasets like Jedit-4.3 is quite visible.

4.3. RQ3: How different feature sets affect the performance of GIS?

Different metric sets were used when comparing the performance of GIS with those of CPDP and WPDP counterparts. In the first case, all features present in the datasets were used to train and test the models. The number of available features in this case are 20, consisting of SCM, OO and LOC metrics. The second case was used in the original study and includes seven features and a subset of OO+LOC, namely CK+LOC. The good performance of these two group (OO and LOC) are reported by multiple studies [1, 36]. In the third and final case, iterative IG subsetting was used to select the most informative set of features in the datasets. These three sets of features were used to train and test all the models in this study, including GIS and CPDP and WPDP benchmarks.

We perform a separate test for each dataset to detect differences among GIS variants. The italic style font is used to represent the best result for each dataset among GIS versions. According to the results presented in Table 5, GIS_{all} outperforms other GIS counterparts in only three cases. GIS_{ckloc} shows better performance in 14 cases and GIS_{IG} achieves the highest among GIS in 22 cases. The KW-H tests do not show a significant difference for two datasets, namely Poi-2.0 and Synapse-1.2. With G the number of cases are three, 15 and 23 respectively for GIS with all, ckloc and IG metrics. This difference in performance, demonstrate the importance of using a refined set of features when searching for the right set of data in CPDP. The difference between different GIS versions is pointed out by the pairwise statistical tests as well. The test results presented in Table 11 show that GIS_{all} underperforms both $\text{GIS}_{\text{ckloc}}$ (Cohen's $d = \{0.280, 0.366\}$) and GIS_{IG} (Cohen's $d = \{0.413, 0.471\}$) in terms of F1 and G. Among the GIS variants, GIS_{IG} achives the highest median and mean F1 and G values and supersedes $\text{GIS}_{\text{ckloc}}$, the second best GIS variant according to the significant $p - values$ and observed effect sizes (Cohen's $d = \{0.168, 0.134\}$).

The Friedman and Nemenyi tests fail to detect a significant difference between these variants. Despite that, with both F1 and G, all GIS variants belong to the top ranking group for which no significant difference is detected from these tests. Please note that we identified a significant difference when comparing only GIS variants against each other with the Friedman and Nemenyi tests in which bolds out the better performances of GIS_{IG} and $\text{GIS}_{\text{ckloc}}$. Moreover, GIS_{IG} achives the highest average ranks in terms of both F1 and G, consistent with our earlier discussed findings.

A depiction of the performance of different GIS variants is presented in Figure 4. The achieved small effect sizes among GIS groups can be seen this figure considering a very similar pattern observed for them.

Better feature selection techniques coupled with the proposed instance selection approach, i.e. GIS, can lead to better predictions and even outperforms WPDP.

5. Discussion

We used NN-Filter approach in the context of our proposed approach by generating the validation datasets used for guiding the evolutionary instance selection process. While Nearest Neighbor selection has been shown to be useful by other studies [4, 17, 22], the usefulness of it for guiding the genetic algorithm is not guaranteed. Nevertheless, GIS which performs on top of NN instance selection as validation dataset, improves it significantly in terms of both F1 and G. A more useful alternative in this case can be the availability of a small portion of within project data that could be used either as a whole or as a part of a better validation dataset since such a dataset could better guide the process due to its extra similarities to the test dataset. This is one of the potential ways to improve GIS and will be investigated in the future.

Table 12 presents the list of the extracted features from the third case of selected features, i.e. iterative InfoGain subsetting for GIS, NN-Filter and

Naive CPDP approaches. These features are sorted based on their importance according to the respective information gain. Note the presence of features LOC and RFC for every project, two of which belonging to CK+LOC. Of the same set, RFC and LCOM are present for the majority of the projects. This in turn is in line with the findings reported by Hall et al. [1] on the usefulness of OO and LOC feature subsets. The performances of these feature sets however, are not as good as they are for GIS with NN-Filter and Naive CPDP and one can see the positive effect of optimization techniques such as our proposed approach in practice.

Please keep in mind that the Nemenyi test is well known to be conservative and usually achieving significance through such tests is difficult. As pointed out in [53], these tests sometimes even fail to detect a significant difference between the best and worst performing groups even though such differences might exist in practice. So the failure to detect such a difference in performance can sometimes be linked to the limited power of tests of this kind.

The ranking procedure used by Nemenyi test could also be problematic. The ranking does not differentiate between a good performing approach that has a slightly lower performance among the benchmarks, on one hand, and an absolute worst performing approach, not even close to the other benchmarks in terms of performance on the other hand. Hence, the decision between a good and a bad approach becomes more difficult (e.g the performances observed from the benchmarks for Poi-3.0 and JEdit-4.3 datasets). Such differences however are considered when the effect size is calculated as in the case for the Wilcoxon tests and Cohen's d values. The two way of comparing the results, i.e. the pairwise tests-effect sizes and the Friedman-Nemenyi tests are chosen according to the aforementioned points.

This however is not to justify the bad performances seen for datasets such as JEdit-4.3, Camel-1.0 and Ivy-1.4 which have bad performances in all benchmarks. One could speculate on the reasons for the bad performances by considering the defect density for these datasets (2.2% for JEdit-4.3, 3.8% for Camel-1.0 and 6.6% for Ivy-1.4). These datasets, usually suffer from a severe case of class imbalance problem, an issue which despite being investigated extensively [14, 15, 16, 17, 19], still seems to be a challenge for CPDP. A step toward solving these problems would be extending/proposing smarter methods/approaches to deal with such problems based on various other distributional characteristics their data.

Finally, our results show the effect of specialized data on performance, selected and refined according to a defined set of criteria. The results not only showed that all the data are not useful in practice, but also considered the data quality issue present in defect prediction data due to their time dependent nature. Such improvements of course might come at a cost of losing one criterion to some extent (such as precision in our experiments) with the benefit of achieving significantly better performance toward other criteria (like recall, F1 and G in the context of our study). Despite that, the achieved results provide the evidence for the usefulness of our proposed approach.

5.1. Runtime

GIS works by generating and evaluating evolving datasets using a search based approach. Consequently, one could expect higher runtime than the conventional models, i.e. feeding the data into a learner after few preprocessing steps and make predictions.

Our goal at this stage was to optimize the effectiveness of CPDP. However, a brief demonstration of the runtime of the approach would be beneficial. As pointed out earlier, the experiments were implemented in Java and Weka library. The spent time for each iteration of each variant was captured for GIS.

The GIS experiments took 1698 minutes (approximately 28.3 hours) in total to complete. This amount of time is spent on performing *30 iterations*41 datasets*3 variants = 3690* runs for the GIS variants.

A rough estimate shows that each GIS iteration requires $1698 \div 3690 \approx 27.6$ seconds. The GIS_{IG} is the fastest of the three, due to the use of customized feature sets. The spent time on average for the datasets in this group is 11.8 seconds with standard deviation of 6.3 seconds. GIS_{all} requires the highest time to finish. The (avg, std) pair for GIS_{all} and $\text{GIS}_{\text{ckloc}}$ are (45.9, 42.7) and (25.09, 44.80) respectively. The high deviations in both cases are caused by releases belonging to camel project (and xalan to some extent in the case of GIS_{all}). These releases have the highest number of instances. Therefore, NN-Filter generated validation datasets would potentially have much more instances, requiring more time for training and testing candidate training datasets for multiple generations and multiple iterations for each dataset.

The mentioned times could be decreased greatly by writing the code in a parallel manner. We ran our experiments in a single thread in a Laptop PC with a core i7 CPU and 8 GB of ram.

6. Threats to Validity

During an empirical study, one should be aware of the potential threats to the validity of the obtained results and derived conclusions [56]. The potential threats to the validity identified for this study are assessed in three categories, namely: construct, external and conclusion validity.

6.1. Construct validity

The metrics used in this study are SCM, OO and LOC which are the only metrics present in the datasets. These metrics have been widely used in previous studies [1, 2, 46, 57]. Even though these metrics can achieve good performances [57], the usefulness of this metrics has been widely criticised [1, 3, 4]. The experimental datasets are collected by Jureczko et al. [31, 32], who cautioned that there could be some mistakes in non defective labels as not all the defects had been found (*regex* search through version control commit comments). This may be a potential threat for defect prediction models training and evaluation; on the other hand, this is one of the issues that GIS is designed to account for. We did not test for different values of k in NN-Filter, but for large datasets, even though only unique elements are selected, the size of the training datasets for NN-Filter could become large. One could expect performance changes depending on different values of k. However, this impact could be for better or worse as seen with some of the datasets for which the Naive CPDP that is trained with all the data lead to better prediction results than NN-Filter with k=10.

6.2. External validity

It is difficult to draw general conclusions from empirical studies of software engineering and our results are limited to the analyzed data and context [58]. Even though many researchers have used subsets of our utilized datasets as the basis of their conclusions, there is no assurance about the generalization of conclusions drawn from these projects. Particularly the applicability of the conclusions for commercial, proprietary and closed source software might be different as there usually are more rigorous code quality standard associated with such projects. Further, all the projects contributing to our study are written in Java and including projects written in other languages surely would affect the generalizability of our findings. On the other hand, in this study we considered a much larger collection of datasets and further investigated and validated some of our findings from our original study. Hence, this study acts not only as an extension to our original study, but also replicates it as well while presenting more evidence for the usefulness of GIS. Having said that, we should note that the external validity threats are usually strong with defect prediction studies and neglecting such threats will bias the conclusions highly.

6.3. Conclusion validity

Our experiments are repeated 30 times to address the randomness and the results are compared using multiple tests, i.e. Kruskal-Wallis H, Friedman and Nemenyi's post-hoc as well as pairwise Wilcoxon signed rank tests. KW-H test requires further post hoc tests to identify the position(s) of detected differences in multiple groups. Since KW-H tests are performed only for individual datasets, we did not perform such post hoc tests as they would have made the analysis very complicated and we decided to select the group with the highest median as the best treatment for that particular dataset whenever a significant $p-value$ is observed from KW-H. However, for Friedman test which is used to compare the overall performance, we used the Nemenyi post-hoc test and presented the results. Further, we performed pairwise Wilcoxon tests to detect possible differences between various GIS versions and other CPDP and WPDP benchmarks. Moreover, to calculate the magnitude of the differences, Cohen's d for related samples was used as effect size. Another threat is the choice of the evaluation measure. Other researchers might consider different measures to evaluate the methods and as a consequence, some of the observations and conclusions may change. Even though our method works better for a large set portion of the datasets (compared with both WPDP and CPDP benchmarks), it is not necessarily better for all of them and further investigation is required.

7. Conclusions

In this study, we further investigated the usefulness of a search based approach to instance selection, i.e., GIS, in the context of cross project defect prediction. Through an evolutionary process, we aimed to converge to an optimal training dataset and at the same time, we considered the effect of feature selection and the potential noise in the labeling of the datasets. We incorporated (NN)-Filter into the model by using it in generating the validation set to optimize the performance of our approach. We generated further refined datasets by utilizing iterative info gain feature subsetting for feature selection. The proposed method outperforms cross project benchmarks significantly in terms of both F1 and G and the achieved large effect sizes. The performance of GIS is also comparable to within project benchmarks. Specifically, GIS outperforms PR while achieving a tie with cross validation. In terms of the effect of feature selection on GIS, we observe that using simple feature selection techniques improves the effectiveness of GIS significantly in comparison with other GIS variants, especially GIS using all features.

Based on the results of this study, we show the usefulness of third party project data and the search based methods in the context of cross project defect prediction. We observed that the performance of a simple classifier like Naive Bayes could be boosted with such approaches. Using a different fitness function targeting other measures like precision, AUC (Area Under the Curve) or other measures may lead to different results while giving the practitioners the flexibility of guiding the process toward their desired goals.

Other validation dataset selection techniques using approaches like clustering, distributional characteristics, small portions of within project data, better and more powerful feature selection techniques and tuning the parameters of the genetic model in addition to designing other fitness functions with a focus on different measures are among possible future works to pursue.

References

[1] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic literature review on fault prediction performance in software engineering, IEEE Trans. Softw. Eng. 38 (6) (2012) 1276–1304. doi:10.1109/TSE.2011.103.
URL http://dx.doi.org/10.1109/TSE.2011.103

[2] M. D'Ambros, M. Lanza, R. Robbes, Evaluating defect prediction approaches: a benchmark and an extensive comparison, Empirical Software Engineering 17 (4-5) (2012) 531–577.

[3] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, Software Engineering, IEEE Transactions on 33 (1) (2007) 2–13.

[4] B. Turhan, T. Menzies, A. B. Bener, J. Di Stefano, On the relative value of cross-company and within-company data for defect prediction, Empirical Software Engineering 14 (5) (2009) 540–578.

[5] V. R. Basili, L. C. Briand, W. L. Melo, A validation of object-oriented design metrics as quality indicators, Software Engineering, IEEE Transactions on 22 (10) (1996) 751–761.

[6] K. El Emam, W. Melo, J. C. Machado, The prediction of faulty classes using object-oriented design metrics, Journal of Systems and Software 56 (1) (2001) 63–75.

[7] T. Gyimothy, R. Ferenc, I. Siket, Empirical validation of object-oriented metrics on open source software for fault prediction, Software Engineering, IEEE Transactions on 31 (10) (2005) 897–910.

[8] N. Nagappan, T. Ball, Static analysis tools as early indicators of pre-release defect density, in: Proceedings of the 27th international conference on Software engineering, ACM, 2005, pp. 580–586.

[9] N. Nagappan, T. Ball, Use of relative code churn measures to predict system defect density, in: Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on, IEEE, 2005, pp. 284–292.

[10] N. Nagappan, T. Ball, A. Zeller, Mining metrics to predict component failures, in: Proceedings of the 28th international conference on Software engineering, ACM, 2006, pp. 452–461.

[11] R. Subramanyam, M. S. Krishnan, Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects, Software Engineering, IEEE Transactions on 29 (4) (2003) 297–310.

[12] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, B. Turhan, The promise repository of empirical software engineering data, West Virginia University, Department of Computer Science.

[13] Z. He, F. Shu, Y. Yang, M. Li, Q. Wang, An investigation on the feasibility of cross-project defect prediction, Automated Software Engineering 19 (2) (2012) 167–199.

[14] L. Chen, B. Fang, Z. Shang, Y. Tang, Negative samples reduction in cross-company software defects prediction, Information and Software Technology 62 (2015) 67–77.

[15] D. Ryu, O. Choi, J. Baik, Value-cognitive boosting with a support vector machine for cross-project defect prediction, Empirical Software Engineering 21 (1) (2016) 43–71.

[16] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, A. E. Hassan, Studying just-in-time defect prediction using cross-project models, Empirical Software Engineering (2015) 1–35.

[17] D. Ryu, J.-I. Jang, J. Baik, A transfer cost-sensitive boosting approach for cross-project defect prediction, Software Quality Journal (2015) 1–38.

[18] S. Herbold, Training data selection for cross-project defect prediction, in: Proceedings of the 9th International Conference on Predictive Models in Software Engineering, ACM, 2013, p. 6.

[19] Y. Ma, G. Luo, X. Zeng, A. Chen, Transfer learning for cross-company software defect prediction, Information and Software Technology 54 (3) (2012) 248–256.

[20] X. Jing, F. Wu, X. Dong, F. Qi, 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, ACM, 2015, pp. 496–507.

[21] B. Turhan, A. T. Mısırlı, A. Bener, Empirical evaluation of the effects of mixed project data on learning defect predictors, Information and Software Technology 55 (6) (2013) 1101–1118.

[22] D. Ryu, J.-I. Jang, J. Baik, A hybrid instance selection using nearest-neighbor for cross-project defect prediction, Journal of Computer Science and Technology 30 (5) (2015) 969–980.

[23] Y. Zhang, D. Lo, X. Xia, 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, IEEE, 2015, pp. 264–269.

[24] S. Uchigaki, S. Uchida, K. Toda, A. Monden, An ensemble approach of simple regression models to cross-project fault prediction, in: Software Engineering, Artificial Intelligence, Networking and Parallel & Dis-

tributed Computing (SNPD), 2012 13th ACIS International Conference on, IEEE, 2012, pp. 476–481.

[25] A. Panichella, R. Oliveto, 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, IEEE, 2014, pp. 164–173.

[26] P. He, B. Li, X. Liu, J. Chen, Y. Ma, An empirical study on software defect prediction with a simplified metric set, Information and Software Technology 59 (2015) 170–190.

[27] L. Yu, A. Mishra, Experience in predicting fault-prone software modules using complexity metrics, Quality Technology & Quantitative Management 9 (4) (2012) 421–434.

[28] J. Nam, S. Kim, Heterogeneous defect prediction, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, 2015, pp. 508–519.

[29] O. Mizuno, Y. Hirata, A cross-project evaluation of text-based fault-prone module prediction, in: Empirical Software Engineering in Practice (IWESEP), 2014 6th International Workshop on, IEEE, 2014, pp. 43–48.

[30] B. Turhan, On the dataset shift problem in software engineering prediction models, Empirical Software Engineering 17 (1-2) (2012) 62–74.

[31] M. Jureczko, L. Madeyski, Towards identifying software project clusters with regard to defect prediction, in: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ACM, 2010, p. 9.

[32] M. Jureczko, D. Spinellis, Using object-oriented design metrics to predict software defects, Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej (2010) 69–81.

[33] J. Śliwerski, T. Zimmermann, A. Zeller, When do changes induce fixes?, SIGSOFT Softw. Eng. Notes 30 (4) (2005) 1–5. doi:10.1145/1082983.1083147.
URL http://doi.acm.org/10.1145/1082983.1083147

[34] M. J. Shepperd, Q. Song, Z. Sun, C. Mair, Data quality: Some comments on the NASA software defect datasets, IEEE Trans. Software Eng. 39 (9) (2013) 1208–1215. `doi:10.1109/TSE.2013.11`.
URL `http://dx.doi.org/10.1109/TSE.2013.11`

[35] S. Hosseini, B. Turhan, M. Mäntylä, Search based training data selection for cross project defect prediction, in: Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2016, ACM, New York, NY, USA, 2016, pp. 3:1–3:10. `doi:10.1145/2972958.2972964`.
URL `http://doi.acm.org/10.1145/2972958.2972964`

[36] Y. C. Liu, T. M. Khoshgoftaar, N. Seliya, Evolutionary optimization of software quality modeling with multiple repositories, Software Engineering, IEEE Transactions on 36 (6) (2010) 852–864.

[37] S. Watanabe, H. Kaiya, K. Kaijiri, Adapting a fault prediction model to allow inter languagereuse, in: Proceedings of the 4th international workshop on Predictor models in software engineering, ACM, 2008, pp. 19–24.

[38] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, 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, ACM, 2009, pp. 91–100.

[39] F. Zhang, A. Mockus, I. Keivanloo, Y. Zou, Towards building a universal defect prediction model with rank transformed predictors, Empirical Software Engineering (2015) 1–39.

[40] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, S. Panichella, Defect prediction as a multiobjective optimization problem, Software Testing, Verification and Reliability 25 (4) (2015) 426–459.

[41] S. J. P. N. N. X. Xia, D. Lo, X. Wang, Hydra: Massively compositional model for cross-project defect prediction., Software Engineering, IEEE Transactions on.

[42] P. He, B. Li, D. Zhang, Y. Ma, Simplification of training data for cross-project defect prediction, arXiv preprint arXiv:1405.0773.

[43] G. Liebchen, M. Shepperd, Data sets and data quality in software engineering: Eight years on, in: Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2016, ACM, New York, NY, USA, 2016, pp. 7:1–7:4. doi:10.1145/2972958.2972967.
URL http://doi.acm.org/10.1145/2972958.2972967

[44] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE transactions on software engineering 33 (1) (2007) 2–13.

[45] Z. He, F. Peters, T. Menzies, Y. Yang, Learning from open-source projects: An empirical study on defect prediction, in: 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, IEEE, 2013, pp. 45–54.

[46] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, Information Sciences 179 (8) (2009) 1040–1058.

[47] B. Turhan, A. Bener, Analysis of naive bayes' assumptions on software fault data: An empirical study, Data & Knowledge Engineering 68 (2) (2009) 278–290.

[48] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, Y. Jiang, Implications of ceiling effects in defect predictors, in: Proceedings of the 4th international workshop on Predictor models in software engineering, ACM, 2008, pp. 47–54.

[49] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, Software Engineering, IEEE Transactions on 34 (4) (2008) 485–496.

[50] S. Sawilowsky, New effect size rules of thumb, Journal of Modern Applied Statistical Methods 8 (2) (2009) 597–599.

[51] V. B. Kampenes, T. Dybå, J. E. Hannay, D. I. Sjøberg, A systematic review of effect size in software engineering experiments, Information and Software Technology 49 (11) (2007) 1073–1086.

[52] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, Journal of the american statistical association 32 (200) (1937) 675–701.

[53] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine learning research 7 (Jan) (2006) 1–30.

[54] P. Nemenyi, Distribution-free multiple comparisons, in: Biometrics, Vol. 18, INTERNATIONAL BIOMETRIC SOC 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210, 1962, p. 263.

[55] J. L. Hintze, R. D. Nelson, Violin plots: a box plot-density trace synergism, The American Statistician 52 (2) (1998) 181–184.

[56] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.

[57] D. Radjenović, M. Heričko, R. Torkar, A. Živkovič, Software fault prediction metrics: A systematic literature review, Information and Software Technology 55 (8) (2013) 1397–1418.

[58] V. R. Basili, F. Shull, F. Lanubile, Building knowledge through families of experiments, Software Engineering, IEEE Transactions on 25 (4) (1999) 456–473.