

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

Seyedrebrvar 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 ($p = \text{values} \ll 0.001$, Cohen's $d = \{0.621, 0.845, 0.762\}$) and G ($p = \text{values} \ll 0.001$, Cohen's $d = \{0.899, 1.114, 1.056\}$), and Naive CPDP (all,ckloc,IG) in terms of F1 ($p = \text{values} \ll 0.001$, Cohen's $d = \{0.743, 0.865, 0.789\}$) and G ($p =$

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

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

22 created on the data from other projects, especially for those with limited or
23 no defect data repository [3, 4, 13].

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

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

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

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

- 59 • Including a more comprehensive set of datasets. In the original study,
60 only the last available dataset from multi-version software projects were
61 used in our experiments. Overall, 13 datasets from PROMISE reposi-
62 tory were used to assess the performance of our proposed approach.
63 We extend the number of datasets to 41 from 11 projects. The reason
64 for the choosing these datasets is their maturity (multiple versions) and
65 the choice of WPDP previous releases benchmark.
- 66 • Including an extra within project benchmark (previous releases). We
67 used a single WPDP benchmark in our original study namely, 10 fold
68 stratified cross validation. Since multiple releases are available for each
69 project in this extension, we compare the performance of GIS with this
70 WPDP benchmark as well.
- 71 • Investigating the effect of different sets of metrics and the sensitivity of
72 GIS toward them. Chidamber & Kemerer (CK)+LOC was used in the
73 original study, due to its reported good performances by Hall et al.[1] as
74 well as recent CPDP studies [25, 36]. To have a better understanding
75 of how GIS reacts to the choice of metrics we used all features and
76 one feature selection technique, i.e. iterative infogain subsetting. We
77 conducted these sets of experiments to address one of the threats to
78 the validity of our original conclusions, i.e. the selection of software
79 metrics. Interestingly, adding SCM to CK+LOC has a positive effect
80 on the WPDP performance despite its negative effect on GIS, therefore
81 making it necessary to be included as failing to so, would be a threat
82 to the conclusion validity.
- 83 • Presenting a more extensive analysis, related work and discussions. A
84 wider range of datasets, benchmarks and results, requires more com-
85 prehensive analysis. The results are presented through multiple types
86 of diagrams (violin plots, critical difference (CD) diagrams, line plots)
87 and the approaches are compared with different targets in mind through
88 statistical tests for pairwise and multiple comparisons for exploring dif-
89 ferent perspectives of the achieved results.

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

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

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

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

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

105 2. Related Work

106 Cross project defect prediction (CPDP) has drawn a great deal of interest
107 recently. To predict defects in projects without sufficient training data, many
108 researchers attempted to build novel and competitive CPDP models [4, 13,
109 18, 21, 31, 37, 38]. However, not all studies report good performances of
110 CPDP [4, 16, 38].

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

118 Zimmermann et al. [38] performed a large scale set of CPDP experiments
119 by creating 622 pair-wise prediction models on 28 datasets from 12 projects
120 (open source and commercial) and observed only 21 pairs (3.4%) that match
121 their performance criteria (precision, recall and accuracy, all greater than
122 0.75). This observation suggests that the majority of predictions will proba-
123 bly fail if training data is not selected carefully. They also found that CPDP
124 is not symmetrical as data from Firefox can predict Internet Explorer defects,
125 but the opposite does not hold. They argued that characteristics of data and
126 process are crucial factors for CPDP.

127 He et al. [13] proposed to use the distributional characteristics (median,
128 mean, variance, standard deviation, skewness, quantiles, etc.) for training
129 dataset selection. They concluded that in the best cases cross project data

130 may provide acceptable prediction results. They also state that training data
131 from the same project does not always lead to better predictions and carefully
132 selected cross project data may provide better prediction results than within-
133 project (WP) data. They also found that data distributional characteristics
134 are informative for training data selection. They used a metalearner built
135 on top of the prediction results of the decision table learner to predict the
136 outcome of the models before making actual predictions.

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

147 Turhan et al. [21] evaluated the effects of mixed project data on predic-
148 tions. They tested whether mixed WP and CP data improves the predic-
149 tion performances. They performed their experiments on 73 versions of 41
150 projects using Naïve Bayes classifier. They concluded that the mixed project
151 data would significantly improve the performance of the defect predictors.

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

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

165 He et al. [26] considered CPDP from the viewpoint of metrics and features
166 by investigating the usefulness of simplified metric sets. They used a greedy
167 approach to filter the list of available metrics and proposed to use different

168 sets of metrics according to the defined criteria. They observed that minimum
169 feature subsets and TOPK metrics could provide acceptable results compared
170 with their benchmarks. They further concluded that the minimum feature
171 subset can improve the predictions despite the acceptable loss of precision.

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

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

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

198 **3. Research Methodology**

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

203 *3.1. Motivation*

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

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

221
222 **Data Quality.** Another inspiration for this work is the fact that the pub-
223 lic datasets are prone to quality issues and contain noisy data [34, 35, 43].
224 Since defects are discovered over time, certain defects might not have been
225 discovered at the time of compiling the datasets and hence, some of the
226 instances in the training set may be misrepresenting themselves as non-
227 defective, while with similar kind of measurements defects can exist in the
228 test set. In contrast, while some test instances are not defective, the most
229 similar items in the training set might be labeled as defective. In short,
230 some of the instances in the test set can have similar measurements with
231 the training set, yet different class labels. Please note that mislabeling may
232 not be the only reason for such situations, and they can occur naturally, i.e.
233 the class labels of similar measurements can differ depending on the metric
234 set used. The acknowledgment of noise in the data and guiding the learning
235 algorithm to account for that can lead to better predictions, as we proposed
236 in our original paper [35] and validate it in this study.

237
238 **Features.** We used CK+LOC metric set originally to assess the per-
239 formance of GIS and the other benchmarks. Hall et al. [1] asserted that
240 OO (Object-Oriented) and LOC have acceptable prediction power and they

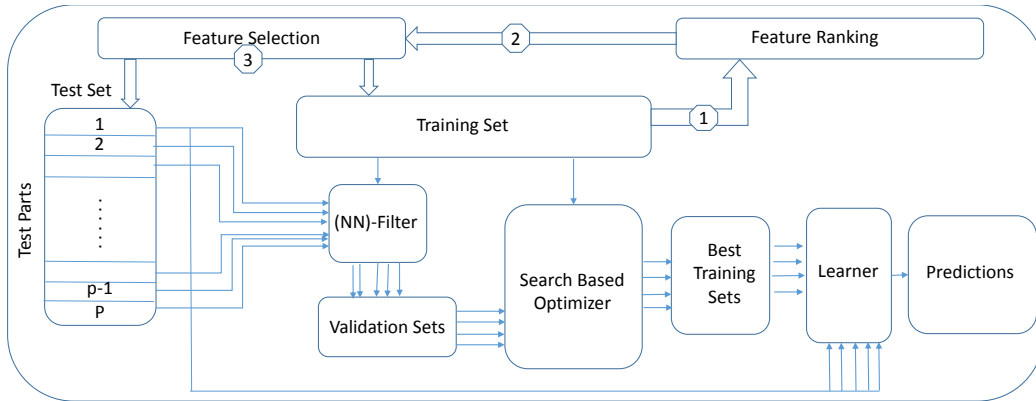


Figure 1: Summary of the search based training data instance selection and performance evaluation process used in this paper.

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

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

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

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

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

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

Algorithm 1 Pseudo code for GIS

```
1: Set numGens = The number of generations of each genetic optimization run.
2: Set popSize = The size of the population.
3: Set DATA = 41 releases from {Ant, Camel, ivy, jedit, log4j, lucene, poi, synapse, velocity, xalan,xerces}
4: Set FEATURES = {CKLOC, All, IG}
5:
6: for FSET in FEATURES do
7:   for RELEASE in DATA do
8:     set TEST = Load Instances from RELEASE with metric set FSET
9:     set TRAIN = Instances from all other projects with metric set FSET
10:    tdSize = 0.02 * Number of instances in TRAIN
11:    for i = 1 to 30 do
12:      Set TestParts = Split TEST instances into p parts
13:
14:      for each testPart in TestParts do
15:        Set vSet = Generate a validation dataset using (NN)-Filter method (with three distance measures).
16:        Set TrainDataSets = Create popSize dataset from TRAIN with replacement each with tdSize instances
17:
18:        for each td in TrainDataSets do
19:          Evaluate td on vSet and add it to the initial generation
20:        end for
21:
22:        for g in range(numGens) do
23:          Create a new generation using the defined Crossover and Mutation function and Elites from the
24:          current generation.
25:          Combine the two generations and extract a new generation
26:        end for
27:
28:        Set bestDS = Select the top dataset from the GA's last iteration.
29:        Evaluate bestDS on testPart and append the results to the pool of results.
30:      end for
31:
32:      Calculate Precision, Recall and F1 and G from the predictions.
33:    end for
34:  end for
35: end for
```

259 features of the datasets are ranked from the highest to the lowest amount of
260 entropy explained. We used iterative InfoGain subsetting [44] to select the
261 appropriate set of features for our experiments. Iterative InfoGain subsetting
262 starts by training the predictors using the top n ranked attributes for $n \in$
263 $\{1, 2, \dots\}$ and continues until a point that having $j + 1$ attributes instead
264 of j does not improve the predictions. An improvement in predictions was
265 measured using F1 values achieved from a 1×10 fold cross validation on
266 the training dataset. The train test splits were identical when adding the
267 features iteratively during the feature selection operation.

268 3.2. Proposed Approach

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

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

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

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

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

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

306
307 **Initial Population:** The initial population is generated using the ran-
308 dom instance selection process with replacement from a large pool of in-
309 stances containing all elements from other projects than the test project.
310 The instances might contain elements included in the validation dataset as

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

Table 1: Chromosome structure

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

315

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

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

329

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

333

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

337

338 **Stopping Criteria:** We place two limitations on the number of itera-
339 tions that the genetic algorithm could progress. The first one is the maximum

340 number of generations allowed. In this case, this number was set to 20. The
341 reason for selecting a relatively small number of generations (20) is due to
342 having small population sizes. The small populations was selected for the
343 runtime considerations. Despite their sizes however, they cover 94.99% of the
344 original training instances on average in every iteration, when creating the
345 initial populations. Further, we observed that the process converges quickly
346 and hence, making 20 an acceptable maximum number of generations. The
347 other stopping criterion is the amount of benefit gained from the population
348 generated. If the difference between the mean fitness of two consecutive pop-
349 ulations is less than $\epsilon = 0.0001$, the genetic algorithm stops. The mentioned
350 epsilon was selected arbitrarily to be a small number. One can expect to
351 achieve better results by tuning these parameters.

352

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

358

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

373

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

Algorithm 2 Mutation

```
1: Input → DS: a dataset
2: Output → A dataset with possible mutated items
3:
4: set mProb = p // Mutation probability
5: set mCount = c // Number of instances to mutate
6: set r = Random value between 0 and 1
7:
8: if r < mProb then
9:   for i in range(mCount) do
10:    Randomly select an instance that is not been mutated in the same round
11:    Find all repeats of the same item and flip their labels
12:   end for
13:
14: end if
```

Algorithm 3 One point crossover

```
1: Input → DS1 and DS2
2: Output → Two new datasets generated from DS1 and DS2
3:
4: Set nDS1 = Empty dataset
5: Set nDS2 = Empty dataset
6: Set point = Random in the range of either of DS1 or DS2
7: SHUFFLE DS1 and DS2
8:
9: for i = 1 to point do
10:  Append DS1(i) to nDS1
11:  Append DS2(i) to nDS2
12: end for
13:
14: for i = point+1 to DS1's length do
15:  Append DS1(i) to nDS2
16:  Append DS2(i) to nDS1
17: end for
18:
19: for each unique instance in nDS1 and nDS2 do
20:  Use the majority voting to decide the label of the instance and its repetitions.
21: end for
```

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

388

389 *3.3. Benchmark Methods*

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

392 **(NN)-Filter (CPDP):** In this approach, the most relevant training in-
393 stances are selected based on a distance measure [4]. In this case, we used
394 10 nearest neighbours and Euclidean distance. This value is similar to that
395 utilized by multiple previous studies [4, 14, 18, 45]. The $k=10$ was the value
396 of choice in the study by Turhan et al. [4] which proposed NN-Filter. The
397 selection of k as 10 was followed by later studies such as He et al. [45] and
398 Chen et al. [14], both of which focus on CPDP. Another CPDP study by
399 Herbold [18] used different values of $k \in \{3, 5, 10, 15, 20, 25, 30\}$ and ob-
400 served the best results for larger k values. The simplicity of the method and
401 the comprehensive number of studies that have tested the approach are the
402 reasons for choosing this method as a benchmark [4, 17, 22]. Also GIS uses
403 (NN)-Filter to select the validation dataset and a benchmark is required to
404 measure the performance difference between (NN)-Filter and GIS.

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

411 **10-Fold cross validation (WPDP):** In this benchmark, we perform
412 stratified cross validation on the test set. Many studies have reported the
413 good or at least better performance of this approach compared with that of
414 cross project methods [4]. Outperforming and improving WPDP is the main
415 goals of many such studies. We refer to this benchmark as **CV** throughout
416 our analysis.

417 **Previous Releases (WPDP):** Previous releases of the same project
418 are used to train the prediction model. Similar to 10-fold cross validation,
419 a good performance of this approach is expected in comparison with that of
420 cross project methods as these older releases are more similar to the test set
421 in comparison with datasets from other projects. More importantly, there is a
422 higher possibility of finding even identical classes in the old and new releases
423 of a project. Previous releases are another target of the CPDP studies as
424 acquiring such data is still difficult in some cases. Note that the first release
425 of each project does not have a previous release and therefore no prediction
426 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

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

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

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

441 3.4. Datasets and Metrics

442 We used 41 releases of 11 projects from the PROMISE repository for our
443 experiments. These projects are open source and all of them have multiple
444 versions. Due to the inclusion of the multi version WPDP benchmark, we
445 skipped the use of datasets with a single version. The datasets are collected
446 by Jureczko, Madeyski and Spinellis [31, 32]. The list of the datasets is
447 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

448 The reason for using these datasets is driven by our goal to account for noise
449 in the data, which is a threat specified by the donors of these datasets. Each
450 dataset contains a number of instances corresponding to the classes in the
451 release. Originally, each instance has 20 static code metrics listed in Table
452 3. Three scenarios were considered for selecting the metric suites. In the
453 first scenario, we used CK+LOC portion of the metrics as the basis of our
454 experiments. CK+LOC is used and validated in previous CPDP studies
455 [2, 46] and Hall et al. [1] have addressed the usefulness of these metrics in
456 comparison with static code metrics. In the second scenario, the full set of
457 metrics were considered for our experiments and finally for the last scenario,
458 we used a very simple and fast feature selection approach based on the rank
459 of the features according to their information gain. The selection of the
460 metrics in our original study was skipped as its primary focus was only on
461 the instance selection problem and using a reduced set that is tried in other
462 studies allowed us to demonstrate the feasibility of our approach as a proof of
463 concept. While this paper includes the same feature set, it also involves the
464 feature selection concept to some extent and detailed analysis are presented
465 accordingly.

466 3.5. Performance Measures and Tools

467 Naïve Bayes (NB) is used as the base learner in all experiments. NB is a
468 member of the probabilistic classifier family that are based on applying Bayes'

469 theorem with strong (naïve) independence assumptions between the features
 470 [47]. The good performance of NB has been shown in many studies. Menzies
 471 et al. [3, 48] and Lessmann et al.[49] have demonstrated the effectiveness
 472 of NB with a set of data mining experiments performed on NASA MDP
 473 datasets. Lessmann et al. compared the most common classifiers on the
 474 NASA datasets and concluded that there is no significant difference between
 475 the performances of top 15 classifiers, one of which is NB [49] .

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

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

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

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

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

484 Using confusion matrix, mentioned indicators are calculated as follows:

485 **Precision:** The proportion of the predicted positive cases that were cor-
 486 rect is calculated using:

$$Precision = \frac{TP}{TP + FP} \quad (3)$$

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

$$Recall = \frac{TP}{TP + FN} \quad (4)$$

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

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (5)$$

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

$$G = \sqrt{precision \times recall} \quad (6)$$

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

503 All the experiments are conducted using **WEKA**¹ machine learning li-
504 brary version 3.6.13. The statistical tests are carried out using the **scipy.stats**²
505 library version 0.16.1, **Python**³ version 3.4.4 and **statistics** library from
506 Python. The violin plots and CD Diagrams are generated using the **mat-**
507 **plotlib**⁴ library version 1.5.3 and **evaluation** package from **Orange**⁵ library
508 version 3.3.8 respectively. A replication package is available online for GIS⁶.

509 4. Results

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

522 These results are depicted through diagrams and plots in Figures 4 and 13.
523 The rankings in the first figure plots are based on the median and critical
524 difference scheme. The third figure provides per datasets results for GIS,
525 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

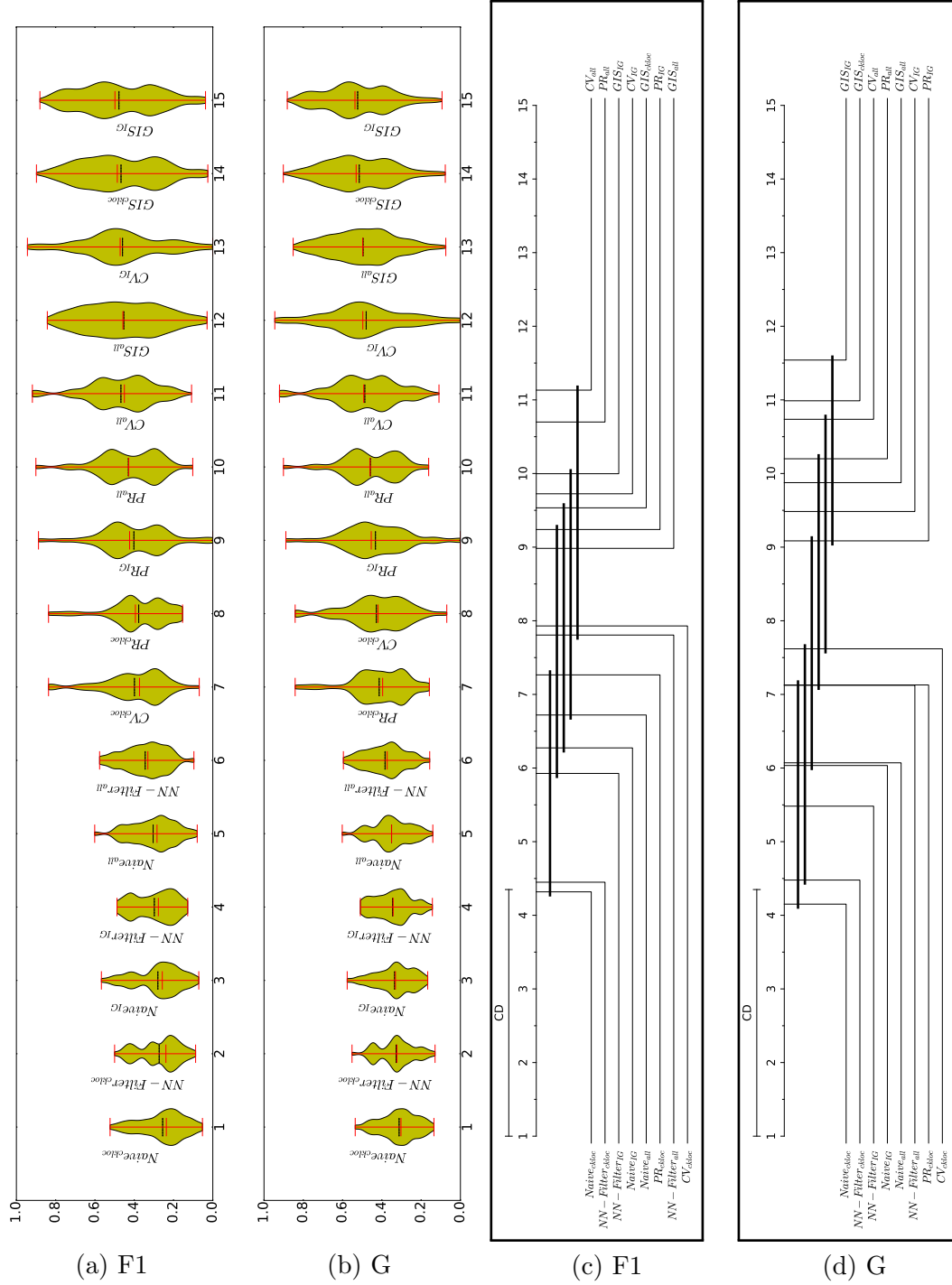


Table 5: F1: GIS vs Cross Project Benchmarks

file	GIS(C)			NN-Filter(C)			Naive(C)		
	All	ckloc	IG	All	ckloc	IG	All	ckloc	IG
ant-1.3	0.292	<i>0.326</i>	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	<i>0.251</i>	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	<i>0.416</i>	0.497	0.424	0.485	0.465	0.437	0.504
camel-1.0	<i>0.102</i>	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	<i>0.173</i>	0.318	0.129	0.182	0.300	0.273	0.292
ivy-2.0	0.277	0.300	<i>0.338</i>	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	<i>0.463</i>	0.422	0.302	0.368	0.468	0.468	0.500
jedit-4.1	0.486	0.486	<i>0.520</i>	0.493	0.414	0.403	0.601	0.523	0.567
jedit-4.2	0.300	0.259	<i>0.342</i>	0.443	0.378	0.420	0.460	0.473	0.481
jedit-4.3	0.043	0.034	<i>0.047</i>	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	<i>0.223</i>	0.421	0.311	0.410	0.333	0.276	0.320
synapse-1.1	0.415	<i>0.457</i>	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	<i>0.311</i>	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

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

Table 6: F1: GIS vs Within Project Benchmarks

file	GIS (C)			CV (W)			PR (W)		
	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

534 calculate Cohen's d :

$$d = \frac{X_{gis} - X_b}{std_p} \quad (7)$$

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

Table 7: G: GIS vs Cross Project Benchmarks

file	GIS(C)			NN-Filter(C)			Naive(C)		
	All	ckloc	IG	All	ckloc	IG	All	ckloc	IG
ant-1.3	0.381	<i>0.434</i>	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	<i>0.359</i>	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	<i>0.501</i>	0.502	0.449	0.504	0.477	0.454	0.523
camel-1.0	<i>0.208</i>	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	<i>0.416</i>	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	<i>0.271</i>	0.331	0.129	0.182	0.306	0.283	0.309
ivy-2.0	0.366	0.391	<i>0.419</i>	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	<i>0.561</i>	0.501	0.444	0.427	0.602	0.536	0.576
jedit-4.2	0.389	0.364	<i>0.435</i>	0.453	0.379	0.420	0.484	0.477	0.484
jedit-4.3	0.114	0.098	<i>0.129</i>	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	<i>0.343</i>	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	<i>0.428</i>	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

538 can be calculated as follows:

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

539 Where s_{gis} , n_{gis} , s_b and n_b are the standard deviation of GIS measure-
540 ments, the number of subjects in the GIS group, standard deviation of
541 benchmark group and number of subjects in the benchmark group, respec-
542 tively. Cohen's d is a way of representing the standardized difference between

Table 8: G: GIS vs Within Project Benchmarks

file	GIS (C)			CV (W)			PR (W)		
	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

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

548 Please note that the measurements are copied multiple times in order
549 to have comparable groups for comparisons in case of NN-Filter and Naive
550 CPDP as no randomness is involved in their settings. Additionally, the

551 results for the first release of each project in PR benchmark is copied from
 552 the same counterpart in CV. In that case, one might see a slight variation in
 553 the results even though there is no actual randomness involved, something
 554 that we have accounted for, in our analysis.

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

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

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

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

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

577 Acquired $CD = 3.3496$ depends on $q_{\alpha} = 3.39123$ which in turn is depen-
 578 dent to the number of compared groups ($k = 15$) as well as the significance

579 level used for the comparisons ($\alpha = 0.05$). Each two approaches are sig-
580 nificantly different whenever their average ranks differ by at least one *CD*.
581 The Friedman test in conjunction with Nemenyi's test rank the approaches
582 with the highest rank belonging to the best performing approach to the low-
583 est based on their average ranks. The results of these tests are presented
584 through CD diagrams in Figure 4 for F1 and G.

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

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

612 Based on the results achieved, the research questions are answered as fol-
613 lows.

614

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.

	GIS _{all}		GIS _{ckloc}		GIS _{IG}	
	p-value	d	p-value	d	p-value	d
CV _{all}	0.000	-0.102	0.202	-0.004	0.451	0.071
CV _{ckloc}	0.000	0.319	0.000	0.413	0.000	0.472
CV _{IG}	0.375	-0.046	0.073	0.055	0.000	0.136
NN-Filter _{all}	0.000	0.522	0.000	0.590	0.000	0.621
NN-Filter _{ckloc}	0.000	0.760	0.000	0.817	0.000	0.845
NN-Filter _{IG}	0.000	0.668	0.000	0.726	0.000	0.762
Naive _{all}	0.000	0.647	0.000	0.700	0.000	0.743
Naive _{ckloc}	0.000	0.776	0.000	0.818	0.000	0.865
Naive _{IG}	0.000	0.692	0.000	0.738	0.000	0.789
PR _{all}	0.527	0.117	0.005	0.189	0.000	0.249
PR _{ckloc}	0.000	0.365	0.000	0.436	0.000	0.505
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.

	GIS _{all}		GIS _{ckloc}		GIS _{IG}	
	p-value	d	p-value	d	p-value	d
CV _{all}	0.001	0.075	0.000	0.243	0.000	0.325
CV _{ckloc}	0.000	0.503	0.000	0.665	0.000	0.739
CV _{IG}	0.000	0.131	0.000	0.288	0.000	0.387
NN-Filter _{all}	0.000	0.729	0.000	0.874	0.000	0.899
NN-Filter _{ckloc}	0.000	0.969	0.000	1.095	0.000	1.114
NN-Filter _{IG}	0.000	0.894	0.000	1.024	0.000	1.056
Naive _{all}	0.000	0.867	0.000	0.980	0.000	1.027
Naive _{ckloc}	0.000	0.975	0.000	1.070	0.000	1.119
Naive _{IG}	0.000	0.884	0.000	0.989	0.000	1.050
PR _{all}	0.000	0.232	0.000	0.364	0.000	0.438
PR _{ckloc}	0.000	0.502	0.000	0.640	0.000	0.732
PR _{IG}	0.000	0.331	0.000	0.448	0.000	0.511

Table 11: GIS vs GIS

	GIS _{all} vs. GIS _{ckloc}		GIS _{all} vs. GIS _{IG}		GIS _{ckloc} vs. 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

615 4.1. RQ1: How is the performance of GIS compared with benchmark cross
616 project defect prediction approaches?

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

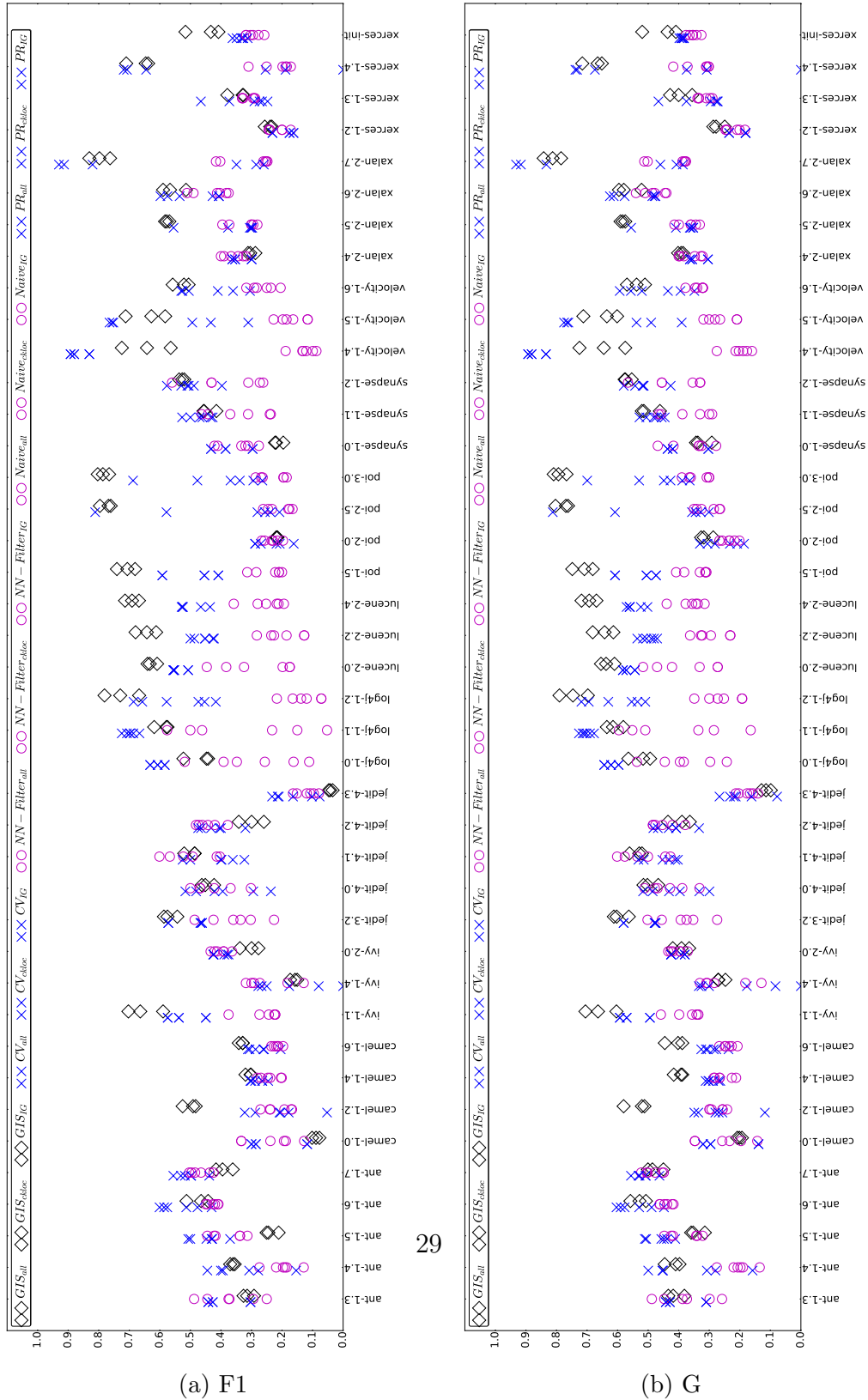
Table 12: Selected features and their order for individual projects for GIS, Naive CPDP and NN-Filter

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

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

627 The violin plots of the measurements for F1 and G values in Figure 4 pro-
628 vide more insights into the results. GIS variants have higher mean, median
629 and max values compared with the CP benchmark methods. More specifi-
630 cally, they provide the first, second and fourth highest median F1 and first,
631 second and third highest G values while the best CPDP benchmark in terms
632 of median F1 and G has the tenth rank. Of course one should note also the
633 weak performances on a couple of datasets and the drop in the minimum
634 value with GIS as well as its wider range of the predictions. From the results
635 in Tables 7 and 5, we can see that GIS has difficulties in predicting datasets
636 like JEdit-4.3 (median F1 around 0.04) and Camel-1.0 (median F1 around
637 0.09). At the same time, the good performances of GIS on datasets from
638 Xalan, Velocity, Synapse and Poi projects to name some, cause a dramatic
639 increase in the max value. Nevertheless, the concentration of the prediction
640 results with GIS is promising, i.e., around half of all predictions are over
641 the maximum values received by the CPDP benchmark methods. Another
642 depiction of the performance difference between GIS and CPDP benchmarks
643 can be seen in Figure 13 with respect to individual datasets. For the ma-
644 jority parts, the location of GIS points are higher than those from CPDP
645 benchmarks while the performance differences in many cases are substantial
646 (e.g for Lucene, Poi, Xalan, Velocity).

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



647 Results of the pairwise statistical tests and the calculated effect sizes show
 648 that GIS is significantly better than both benchmark CPDP approaches and
 649 the effect sizes confirm this conclusion. GIS_{all} which achieves the lowest
 650 performance in the GIS group, outperforms (NN)-Filter (all, ckloc, IG) in
 651 terms of F1 ($p - value \ll 0.001$, Cohen's $d = \{0.522, 0.760, 0.668\}$) and G
 652 ($p - value \ll 0.001$, Cohen's $d = \{0.729, 0.969, 0.894\}$). It also outper-
 653 forms naive CPDP (all, ckloc, IG) in terms of F1 ($p - value \ll 0.001$,
 654 Cohen's $d = \{0.647, 0.776, 0.692\}$) and G ($p - value \ll 0.001$, Cohen's
 655 $d = \{0.0.867, 0.975, 0.884\}$). These performance improvements are more vis-
 656 ible with G and the effect sizes are larger. With careful selection of the
 657 features, GIS could achieve even better results. Iterative IG in GIS_{IG} leads
 658 to better predictions in comparison with (NN)-Filter (best case) by provid-
 659 ing the effect sizes of 0.621 and 0.899 in terms of F1 and G. Similarly, It
 660 outperforms Naive CPDP (best case) with the effect sizes of 0.743 and 1.027
 661 with F1 and G.

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

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

684

685 4.2. RQ2: How is the performance of GIS compared with the within project
686 defect prediction approach?

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

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

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

702 With G, GIS is significantly better than all benchmark WPDP cases,
703 but in some cases the effect sizes are very small. GIS has a tiny difference
704 with CV_{all} (which provides the best WPDP performances considering both
705 median and the range of the values-stability) based on the obtained effect size
706 (0.075). Despite that, higher and significant effect sizes could be observed as
707 well in case of CV_{ckloc} and PR_{ckloc} with effect sizes 0.739 and 0.732 (medium
708 to large) respectively.

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

722 $G=(1+2+5)$ compared with CV which achieves $G=(3+6+8)$. GIS outper-
723 forms PR in terms of rank sums with both $F1=(2+6+10)$ and $G=(4+7+9)$.

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

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

736 *4.3. RQ3: How different feature sets affect the performance of GIS?*

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

748 We perform a separate test for each dataset to detect differences among
749 GIS variants. The italic style font is used to represent the best result for
750 each dataset among GIS versions. According to the results presented in Ta-
751 ble 5, GIS_{all} outperforms other GIS counterparts in only three cases. GIS_{ckloc}
752 shows better performance in 14 cases and GIS_{IG} achieves the highest among
753 GIS in 22 cases. The KW-H tests do not show a significant difference for
754 two datasets, namely Poi-2.0 and Synapse-1.2. With G the number of cases
755 are three, 15 and 23 respectively for GIS with all, ckloc and IG metrics.
756 This difference in performance, demonstrate the importance of using a re-
757 fined set of features when searching for the right set of data in CPDP. The

758 difference between different GIS versions is pointed out by the pairwise sta-
759 tistical tests as well. The test results presented in Table 11 show that GIS_{all}
760 underperforms both GIS_{ckloc} (Cohen’s $d = \{0.280, 0.366\}$) and GIS_{IG} (Co-
761 hen’s $d = \{0.413, 0.471\}$) in terms of F1 and G. Among the GIS variants,
762 GIS_{IG} achieves the highest median and mean F1 and G values and supersedes
763 GIS_{ckloc}, the second best GIS variant according to the significant $p - values$
764 and observed effect sizes (Cohen’s $d = \{0.168, 0.134\}$).

765 The Friedman and Nemenyi tests fail to detect a significant difference
766 between these variants. Despite that, with both F1 and G, all GIS variants
767 belong to the top ranking group for which no significant difference is detected
768 from these tests. Please note that we identified a significant difference when
769 comparing only GIS variants against each other with the Friedman and Ne-
770 menyi tests in which holds out the better performances of GIS_{IG} and GIS_{ckloc}.
771 Moreover, GIS_{IG} achieves the highest average ranks in terms of both F1 and
772 G, consistent with our earlier discussed findings.

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

776 Better feature selection techniques coupled with the proposed instance
777 selection approach, i.e. GIS, can lead to better predictions and even outper-
778 forms WPDP.

779 5. Discussion

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

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

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

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

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

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

830 Finally, our results show the effect of specialized data on performance,

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

840 5.1. Runtime

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

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

850 The GIS experiments took 1698 minutes (approximately 28.3 hours) in to-
851 tal to complete. This amount of time is spent on performing $30 \text{ iterations} * 41$
852 $\text{datasets} * 3 \text{ variants} = 3690$ runs for the GIS variants.

853 A rough estimate shows that each GIS iteration requires $1698 \div 3690 \approx$
854 27.6 seconds. The GIS_{IG} is the fastest of the three, due to the use of cus-
855 tomized feature sets. The spent time on average for the datasets in this group
856 is 11.8 seconds with standard deviation of 6.3 seconds. GIS_{all} requires the
857 highest time to finish. The (avg, std) pair for GIS_{all} and GIS_{ckloc} are (45.9,
858 42.7) and (25.09, 44.80) respectively. The high deviations in both cases are
859 caused by releases belonging to camel project (and xalan to some extent in
860 the case of GIS_{all}). These releases have the highest number of instances.
861 Therefore, NN-Filter generated validation datasets would potentially have
862 much more instances, requiring more time for training and testing candidate
863 training datasets for multiple generations and multiple iterations for each
864 dataset.

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

868 6. Threats to Validity

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

873 6.1. Construct validity

874 The metrics used in this study are SCM, OO and LOC which are the
875 only metrics present in the datasets. These metrics have been widely used in
876 previous studies [1, 2, 46, 57]. Even though these metrics can achieve good
877 performances [57], the usefulness of this metrics has been widely criticised
878 [1, 3, 4]. The experimental datasets are collected by Jureczko et al. [31, 32],
879 who cautioned that there could be some mistakes in non defective labels as
880 not all the defects had been found (*regex* search through version control
881 commit comments). This may be a potential threat for defect prediction
882 models training and evaluation; on the other hand, this is one of the issues
883 that GIS is designed to account for. We did not test for different values of
884 k in NN-Filter, but for large datasets, even though only unique elements are
885 selected, the size of the training datasets for NN-Filter could become large.
886 One could expect performance changes depending on different values of k .
887 However, this impact could be for better or worse as seen with some of the
888 datasets for which the Naive CPDP that is trained with all the data lead to
889 better prediction results than NN-Filter with $k=10$.

890 6.2. External validity

891 It is difficult to draw general conclusions from empirical studies of software
892 engineering and our results are limited to the analyzed data and context [58].
893 Even though many researchers have used subsets of our utilized datasets as
894 the basis of their conclusions, there is no assurance about the generalization
895 of conclusions drawn from these projects. Particularly the applicability of the
896 conclusions for commercial, proprietary and closed source software might be
897 different as there usually are more rigorous code quality standard associated
898 with such projects. Further, all the projects contributing to our study are
899 written in Java and including projects written in other languages surely would
900 affect the generalizability of our findings. On the other hand, in this study
901 we considered a much larger collection of datasets and further investigated
902 and validated some of our findings from our original study. Hence, this study

903 acts not only as an extension to our original study, but also replicates it as
904 well while presenting more evidence for the usefulness of GIS. Having said
905 that, we should note that the external validity threats are usually strong with
906 defect prediction studies and neglecting such threats will bias the conclusions
907 highly.

908 *6.3. Conclusion validity*

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

929 **7. Conclusions**

930 In this study, we further investigated the usefulness of a search based ap-
931 proach to instance selection, i.e., GIS, in the context of cross project defect
932 prediction. Through an evolutionary process, we aimed to converge to an
933 optimal training dataset and at the same time, we considered the effect of
934 feature selection and the potential noise in the labeling of the datasets. We
935 incorporated (NN)-Filter into the model by using it in generating the valida-
936 tion set to optimize the performance of our approach. We generated further
937 refined datasets by utilizing iterative info gain feature subsetting for feature

938 selection. The proposed method outperforms cross project benchmarks sig-
939 nificantly in terms of both F1 and G and the achieved large effect sizes. The
940 performance of GIS is also comparable to within project benchmarks. Specif-
941 ically, GIS outperforms PR while achieving a tie with cross validation. In
942 terms of the effect of feature selection on GIS, we observe that using simple
943 feature selection techniques improves the effectiveness of GIS significantly in
944 comparison with other GIS variants, especially GIS using all features.

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

952 Other validation dataset selection techniques using approaches like clus-
953 tering, distributional characteristics, small portions of within project data,
954 better and more powerful feature selection techniques and tuning the param-
955 eters of the genetic model in addition to designing other fitness functions with
956 a focus on different measures are among possible future works to pursue.

957 **References**

- 958 [1] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A system-
959 atic literature review on fault prediction performance in software en-
960 gineering, *IEEE Trans. Softw. Eng.* 38 (6) (2012) 1276–1304. doi:
961 10.1109/TSE.2011.103.
962 URL <http://dx.doi.org/10.1109/TSE.2011.103>
- 963 [2] M. D'Ambros, M. Lanza, R. Robbes, Evaluating defect prediction ap-
964 proaches: a benchmark and an extensive comparison, *Empirical Soft-
965 ware Engineering* 17 (4-5) (2012) 531–577.
- 966 [3] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes
967 to learn defect predictors, *Software Engineering, IEEE Transactions on*
968 33 (1) (2007) 2–13.
- 969 [4] B. Turhan, T. Menzies, A. B. Bener, J. Di Stefano, On the relative
970 value of cross-company and within-company data for defect prediction,
971 *Empirical Software Engineering* 14 (5) (2009) 540–578.

- 972 [5] V. R. Basili, L. C. Briand, W. L. Melo, A validation of object-oriented
973 design metrics as quality indicators, *Software Engineering, IEEE Trans-*
974 *actions on* 22 (10) (1996) 751–761.
- 975 [6] K. El Emam, W. Melo, J. C. Machado, The prediction of faulty classes
976 using object-oriented design metrics, *Journal of Systems and Software*
977 56 (1) (2001) 63–75.
- 978 [7] T. Gyimothy, R. Ferenc, I. Siket, Empirical validation of object-oriented
979 metrics on open source software for fault prediction, *Software Engineer-*
980 *ing, IEEE Transactions on* 31 (10) (2005) 897–910.
- 981 [8] N. Nagappan, T. Ball, Static analysis tools as early indicators of pre-
982 release defect density, in: *Proceedings of the 27th international confer-*
983 *ence on Software engineering*, ACM, 2005, pp. 580–586.
- 984 [9] N. Nagappan, T. Ball, Use of relative code churn measures to predict
985 system defect density, in: *Software Engineering, 2005. ICSE 2005. Pro-*
986 *ceedings. 27th International Conference on*, IEEE, 2005, pp. 284–292.
- 987 [10] N. Nagappan, T. Ball, A. Zeller, Mining metrics to predict component
988 failures, in: *Proceedings of the 28th international conference on Software*
989 *engineering*, ACM, 2006, pp. 452–461.
- 990 [11] R. Subramanyam, M. S. Krishnan, Empirical analysis of ck metrics
991 for object-oriented design complexity: Implications for software defects,
992 *Software Engineering, IEEE Transactions on* 29 (4) (2003) 297–310.
- 993 [12] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, B. Turhan,
994 The promise repository of empirical software engineering data, West
995 Virginia University, Department of Computer Science.
- 996 [13] Z. He, F. Shu, Y. Yang, M. Li, Q. Wang, An investigation on the feasibil-
997 ity of cross-project defect prediction, *Automated Software Engineering*
998 19 (2) (2012) 167–199.
- 999 [14] L. Chen, B. Fang, Z. Shang, Y. Tang, Negative samples reduction in
1000 cross-company software defects prediction, *Information and Software*
1001 *Technology* 62 (2015) 67–77.

- 1002 [15] D. Ryu, O. Choi, J. Baik, Value-cognitive boosting with a support vec-
1003 tor machine for cross-project defect prediction, *Empirical Software En-*
1004 *gineering* 21 (1) (2016) 43–71.
- 1005 [16] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi,
1006 A. E. Hassan, Studying just-in-time defect prediction using cross-project
1007 models, *Empirical Software Engineering* 21 (5) (2016) 2072–2106.
- 1008 [17] D. Ryu, J.-I. Jang, J. Baik, A transfer cost-sensitive boosting approach
1009 for cross-project defect prediction, *Software Quality Journal* (2015) 1–
1010 38.
- 1011 [18] S. Herbold, Training data selection for cross-project defect prediction, in:
1012 *Proceedings of the 9th International Conference on Predictive Models*
1013 *in Software Engineering*, ACM, 2013, p. 6.
- 1014 [19] Y. Ma, G. Luo, X. Zeng, A. Chen, Transfer learning for cross-company
1015 software defect prediction, *Information and Software Technology* 54 (3)
1016 (2012) 248–256.
- 1017 [20] X. Jing, F. Wu, X. Dong, F. Qi, B. Xu, Heterogeneous cross-company
1018 defect prediction by unified metric representation and cca-based transfer
1019 learning, in: *Proceedings of the 2015 10th Joint Meeting on Foundations*
1020 *of Software Engineering*, ACM, 2015, pp. 496–507.
- 1021 [21] B. Turhan, A. T. Mısırlı, A. Bener, Empirical evaluation of the effects
1022 of mixed project data on learning defect predictors, *Information and*
1023 *Software Technology* 55 (6) (2013) 1101–1118.
- 1024 [22] D. Ryu, J.-I. Jang, J. Baik, A hybrid instance selection using nearest-
1025 neighbor for cross-project defect prediction, *Journal of Computer Sci-*
1026 *ence and Technology* 30 (5) (2015) 969–980.
- 1027 [23] Y. Zhang, D. Lo, X. Xia, J. Sun, An empirical study of classifier com-
1028 bination for cross-project defect prediction, in: *Computer Software and*
1029 *Applications Conference (COMPSAC)*, 2015 IEEE 39th Annual, Vol. 2,
1030 IEEE, 2015, pp. 264–269.
- 1031 [24] S. Uchigaki, S. Uchida, K. Toda, A. Monden, An ensemble approach
1032 of simple regression models to cross-project fault prediction, in: *Soft-*
1033 *ware Engineering, Artificial Intelligence, Networking and Parallel & Dis-*

- 1034 tributed Computing (SNPD), 2012 13th ACIS International Conference
1035 on, IEEE, 2012, pp. 476–481.
- 1036 [25] A. Panichella, R. Oliveto, A. De Lucia, Cross-project defect prediction
1037 models: L’union fait la force, in: Software Maintenance, Reengineering
1038 and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution
1039 Week-IEEE Conference on, IEEE, 2014, pp. 164–173.
- 1040 [26] P. He, B. Li, X. Liu, J. Chen, Y. Ma, An empirical study on software
1041 defect prediction with a simplified metric set, *Information and Software
1042 Technology* 59 (2015) 170–190.
- 1043 [27] L. Yu, A. Mishra, Experience in predicting fault-prone software modules
1044 using complexity metrics, *Quality Technology & Quantitative Manage-
1045 ment* 9 (4) (2012) 421–434.
- 1046 [28] J. Nam, S. Kim, Heterogeneous defect prediction, in: *Proceedings of the
1047 2015 10th Joint Meeting on Foundations of Software Engineering*, ACM,
1048 2015, pp. 508–519.
- 1049 [29] O. Mizuno, Y. Hirata, A cross-project evaluation of text-based fault-
1050 prone module prediction, in: *Empirical Software Engineering in Practice
1051 (IWESEP)*, 2014 6th International Workshop on, IEEE, 2014, pp. 43–48.
- 1052 [30] B. Turhan, On the dataset shift problem in software engineering predic-
1053 tion models, *Empirical Software Engineering* 17 (1-2) (2012) 62–74.
- 1054 [31] M. Jureczko, L. Madeyski, Towards identifying software project clusters
1055 with regard to defect prediction, in: *Proceedings of the 6th International
1056 Conference on Predictive Models in Software Engineering*, ACM, 2010,
1057 p. 9.
- 1058 [32] M. Jureczko, D. Spinellis, Using object-oriented design metrics to predict
1059 software defects, *Models and Methods of System Dependability*. Oficyna
1060 Wydawnicza Politechniki Wrocławskiej (2010) 69–81.
- 1061 [33] J. Śliwerski, T. Zimmermann, A. Zeller, When do changes induce fixes?,
1062 *SIGSOFT Softw. Eng. Notes* 30 (4) (2005) 1–5. doi:10.1145/1082983.
1063 1083147.
1064 URL <http://doi.acm.org/10.1145/1082983.1083147>

- 1065 [34] M. J. Shepperd, Q. Song, Z. Sun, C. Mair, Data quality: Some comments
1066 on the NASA software defect datasets, *IEEE Trans. Software Eng.* 39 (9)
1067 (2013) 1208–1215. doi:10.1109/TSE.2013.11.
1068 URL <http://dx.doi.org/10.1109/TSE.2013.11>
- 1069 [35] S. Hosseini, B. Turhan, M. Mäntylä, Search based training data selection
1070 for cross project defect prediction, in: *Proceedings of the The 12th*
1071 *International Conference on Predictive Models and Data Analytics in*
1072 *Software Engineering, PROMISE 2016*, ACM, New York, NY, USA,
1073 2016, pp. 3:1–3:10. doi:10.1145/2972958.2972964.
1074 URL <http://doi.acm.org/10.1145/2972958.2972964>
- 1075 [36] Y. C. Liu, T. M. Khoshgoftaar, N. Seliya, Evolutionary optimization
1076 of software quality modeling with multiple repositories, *Software Engi-*
1077 *neering, IEEE Transactions on* 36 (6) (2010) 852–864.
- 1078 [37] S. Watanabe, H. Kaiya, K. Kaijiri, Adapting a fault prediction model
1079 to allow inter languagereuse, in: *Proceedings of the 4th international*
1080 *workshop on Predictor models in software engineering*, ACM, 2008, pp.
1081 19–24.
- 1082 [38] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, Cross-
1083 project defect prediction: a large scale experiment on data vs. domain
1084 vs. process, in: *Proceedings of the the 7th joint meeting of the European*
1085 *software engineering conference and the ACM SIGSOFT symposium on*
1086 *The foundations of software engineering*, ACM, 2009, pp. 91–100.
- 1087 [39] F. Zhang, A. Mockus, I. Keivanloo, Y. Zou, Towards building a universal
1088 defect prediction model with rank transformed predictors, *Empirical*
1089 *Software Engineering* (2015) 1–39.
- 1090 [40] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella,
1091 S. Panichella, Defect prediction as a multiobjective optimization prob-
1092 lem, *Software Testing, Verification and Reliability* 25 (4) (2015) 426–459.
- 1093 [41] X. Xia, D. Lo, S. J. Pan, N. Nagappan, X. Wang, Hydra: Massively com-
1094 positional model for cross-project defect prediction, *IEEE Transactions*
1095 *on Software Engineering* 42 (10) (2016) 977–998.
- 1096 [42] P. He, B. Li, D. Zhang, Y. Ma, Simplification of training data for cross-
1097 project defect prediction, arXiv preprint arXiv:1405.0773.

- 1098 [43] G. Liebchen, M. Shepperd, Data sets and data quality in software engi-
1099 neering: Eight years on, in: Proceedings of the The 12th International
1100 Conference on Predictive Models and Data Analytics in Software Engi-
1101 neering, PROMISE 2016, ACM, New York, NY, USA, 2016, pp. 7:1–7:4.
1102 doi:10.1145/2972958.2972967.
1103 URL <http://doi.acm.org/10.1145/2972958.2972967>
- 1104 [44] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes
1105 to learn defect predictors, *IEEE transactions on software engineering*
1106 33 (1) (2007) 2–13.
- 1107 [45] Z. He, F. Peters, T. Menzies, Y. Yang, Learning from open-source
1108 projects: An empirical study on defect prediction, in: 2013 ACM/IEEE
1109 International Symposium on Empirical Software Engineering and Mea-
1110 surement, IEEE, 2013, pp. 45–54.
- 1111 [46] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets,
1112 and feature selection techniques on software fault prediction problem,
1113 *Information Sciences* 179 (8) (2009) 1040–1058.
- 1114 [47] B. Turhan, A. Bener, Analysis of naive bayes’ assumptions on software
1115 fault data: An empirical study, *Data & Knowledge Engineering* 68 (2)
1116 (2009) 278–290.
- 1117 [48] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, Y. Jiang, Im-
1118 plications of ceiling effects in defect predictors, in: Proceedings of the
1119 4th international workshop on Predictor models in software engineering,
1120 ACM, 2008, pp. 47–54.
- 1121 [49] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classi-
1122 fication models for software defect prediction: A proposed framework
1123 and novel findings, *Software Engineering, IEEE Transactions on* 34 (4)
1124 (2008) 485–496.
- 1125 [50] S. Sawilowsky, New effect size rules of thumb, *Journal of Modern Applied*
1126 *Statistical Methods* 8 (2) (2009) 597–599.
- 1127 [51] V. B. Kampenes, T. Dybå, J. E. Hannay, D. I. Sjøberg, A systematic
1128 review of effect size in software engineering experiments, *Information*
1129 *and Software Technology* 49 (11) (2007) 1073–1086.

- 1130 [52] M. Friedman, The use of ranks to avoid the assumption of normality
1131 implicit in the analysis of variance, *Journal of the american statistical*
1132 *association* 32 (200) (1937) 675–701.
- 1133 [53] J. Demšar, Statistical comparisons of classifiers over multiple data sets,
1134 *Journal of Machine learning research* 7 (Jan) (2006) 1–30.
- 1135 [54] P. Nemenyi, Distribution-free multiple comparisons, in: *Biometrics*,
1136 Vol. 18, INTERNATIONAL BIOMETRIC SOC 1441 I ST, NW, SUITE
1137 700, WASHINGTON, DC 20005-2210, 1962, p. 263.
- 1138 [55] J. L. Hintze, R. D. Nelson, Violin plots: a box plot-density trace syner-
1139 gism, *The American Statistician* 52 (2) (1998) 181–184.
- 1140 [56] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén,
1141 *Experimentation in software engineering*, Springer Science & Business
1142 Media, 2012.
- 1143 [57] D. Radjenović, M. Heričko, R. Torkar, A. Živkovič, Software fault predic-
1144 tion metrics: A systematic literature review, *Information and Software*
1145 *Technology* 55 (8) (2013) 1397–1418.
- 1146 [58] V. R. Basili, F. Shull, F. Lanubile, Building knowledge through fami-
1147 lies of experiments, *Software Engineering, IEEE Transactions on* 25 (4)
1148 (1999) 456–473.