Search Based Training Data Selection For Cross Project Defect Prediction

Seyedrebvar Hosseini
M3S, Faculty of Information Technology and Electrical Engineering
University of Oulu
90014, Oulu, Finland
rebvar@oulu.fi

Burak Turhan
M3S, Faculty of Information Technology and Electrical Engineering
University of Oulu
90014, Oulu, Finland
burak.turhan@oulu.fi

Mika Mäntylä
M3S, Faculty of Information Technology and Electrical Engineering
University of Oulu
90014, Oulu, Finland
mika.mantyla@oulu.fi

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, data quality is an issue to consider in CPDP.
Aim: We aim at utilising the Nearest Neighbor (NN)-Filter, embedded in a genetic algorithm, for generating evolving training datasets to tackle CPDP, while accounting for potential noise in defect labels.
Method: We propose a new search based training data (i.e., instance) selection approach for CPDP called GIS (Genetic Instance Selection) that looks for solutions to optimize a combined measure of F-Measure and GMean, on a validation set generated by (NN)-filter. The genetic operations consider the similarities in features and address possible noise in assigned defect labels. We use 13 datasets from PROMISE repository in order to compare the performance of GIS with benchmark CPDP methods, namely (NN)-filter and naive CPDP, as well as with within project defect prediction (WPDP).
Results: Our results show that GIS is significantly better than (NN)-Filter in terms of F-Measure ($p-value \ll 0.001$, Cohen's $d = 0.697$) and GMean ($p-value \ll 0.001$, Cohen's $d = 0.946$). It also outperforms the naive CPDP approach in terms of F-Measure ($p-value \ll 0.001$, Cohen's $d = 0.753$) and GMean ($p-value \ll 0.001$, Cohen's $d = 0.994$). In addition, the performance of our approach is better than that of WPDP, again considering F-Measure ($p-value \ll 0.001$, Cohen's $d = 0.227$) and GMean ($p-value \ll 0.001$, Cohen's $d = 0.595$) values.
Conclusions: We conclude that search based instance selection 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 low precision. Using different optimization goals, e.g. targeting high precision, would be a future direction to investigate.

Keywords

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

1. INTRODUCTION

Software Defect Prediction (SDP) is among the most studied and challenging problems in the field of software engineering [8]. Software testing can be very time consuming while the resources might be limited, hence detecting defects in an automated way can save lots of time and effort [5, 17, 30].

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 [1, 6, 7, 19, 20, 21, 27]. 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 being criticized as within project data is usually not available for new projects. On the other hand, there are plenty of relevant public datasets available, especially in the open source repositories [16]. Using the available public datasets, one can investigate the usefulness of models created on the data from other projects, especially for those with limited or no defect data repository [10, 17, 30].

Learning approach and the training data are two major elements in building high performance prediction models. Finding a suitable set with similar defect distribution characteristics as the test set is likely to increase the performance of prediction models [28]. Since defect detection and label assignment is based on mining version control systems [12, 13, 26], the process could be prone to errors and data quality can be questionable [25]. In other words, the labels of some of the instances might not have been identified correctly, two or more instances with same measurements can have different labels, or undetected defects may not be captured in the dataset.

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

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

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

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

2. RELATED WORK

Cross project defect prediction (CPDP) has drawn a great deal of interest recently. To predict defects in projects without sufficient training data, many researchers attempted to build CPDP models [10, 11, 12, 30, 31, 32, 35]. However, most studies report poor performances for CPDP [30, 35].

Turhan et al. [30] observed that CPDP under-performs WPDP. They also found that despite its good probability of detection rates, CPDP causes excessive false alarms. To overcome this problem, they proposed the (NN)-Filter to select the most relevant training data instances from a pool of cross project datasets. Although this method lowered the false alarm rates dramatically, its performance was still worse than WPDP.

Zimmermann et al. [35] tested the CPDP approach for 622 pairs of 28 datasets from 12 projects (open source and commercial) and found only 21 pairs (3.4%) that match their performance criteria (precision, recall and accuracy, all greater than 0.75). This means that the predictions will fail in most cases if training data is not selected carefully. They also found that CPDP is not symmetrical as data from Firefox can predict Internet Explorer defects, but the opposite does not hold. They argued that characteristics of data and process are crucial factors for CPDP.

He et al. [10] proposed to use the distributional characteristics (median, mean, variance, standard deviation, skewness, quantiles, etc.) for training dataset selection. They conclude that in the best cases cross project data may provide acceptable prediction results. They also state that training data from the same project does not always lead to better predictions and carefully selected cross project data may provide better prediction results than within-project (WP) data. They also found that data distributional characteristics are informative for training data selection.

Herbold [11] proposed distance-based strategies for the selection of training data based on distributional characteristics of the available data. They presented two strategies based on EM (ExpectationâĂŞMaximization) clustering and NN (Nearest Neighbor) algorithm with distributional characteristics as the decision strategy. They evaluated the strategies in a large case study with 44 versions of 14 software projects and they observed that i) weights can be used to successfully deal with biased data and ii) the training data selection provides a significant improvement in the success rate and recall of defect detection. However, their overall success rate was still too low for the practical application of CPDP.

Turhan et al. [31] evaluated the effects of mixed project data on predictions. They tested whether mixed WP and CP data improves the prediction performances. They performed their experiments on 73 versions of 41 projects using Naïve Bayes classifier. They concluded that the mixed project data would significantly improve the performance of the defect predictors.

Zhang et al [34] created a universal defect prediction model from a large pool of 1,385 projects with the aim of relieving the need to build prediction models for individual projects. They approached the problem of variations in the distributions by clustering and rank transformation using the similarities among the projects. Based on their results, their model obtained prediction performance comparable to the WP models when applied to five external projects and performed similarly among projects with different context factors.

Ryu et al. [23] presented a Hybrid Instance Selection with the Nearest Neighbor (HISNN) method using a hybrid classification to address the class imbalance for CPDP. Their approach used a combination of the Nearest Neighbour algorithm with Hamming distance and Naïve Bayes to address the instance selection problem.

While the abovementioned studies focus on the dataset and instance selection problems, none of them are using the search based approach. Search based approaches to defect prediction problem have been considered by Liu et al., who tried to come up with mathematical expressions as their solutions that maximize the effectiveness of their approach [15]. They compared their approach with 17 non-evolutionary machine learning algorithms and concluded that the search-based models decrease the misclassification rate consistently compared with the non-search-based models. In addition Canfora et al. proposed a search based multi-objective optimization approach [3]. Using multi-objective genetic algorithm NSGA-II, they tried to come up with an optimal cost effectiveness model for CPDP. They concluded that their approach outperforms the single objective, trivial and local prediction approaches. These studies are not focused on the instance/dataset selection problem and hence, differ from our approach.

In summary, we combine training data instance selection with search based methods with the purpose of addressing potential defect labeling errors and compare our method's performance with benchmark CPDP and WPDP approaches.

3. RESEARCH METHODOLOGY

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

3.1 Motivation

If selected carefully, a dataset from other projects can provide a better predicting power than WP data [10] as the large pool of the available CP data has the potential to cover a larger range of the feature space. This may lead to a better match between training and test datasets and consequently to better predicting power.

One of the first attempts in this area was the idea of filtering the training dataset instances [30]. In this approach, the most similar instances from a large pool containing all the training instances from other projects are selected using the k-NN algorithm. Using the distributional characteristics of the test and training datasets is another approach used in multiple studies [9, 10]. Clustering the instances

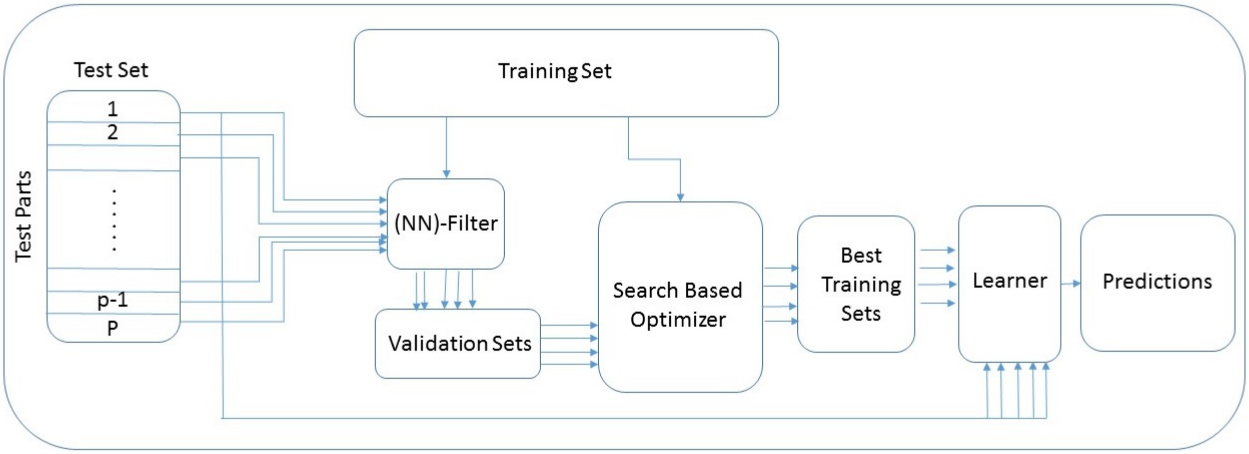


Figure 1: Summary of the search based training data instance selection and performance evaluation process used in this paper.

is yet another approach used in other studies [11, 12, 34]. While these methods have been shown to be useful, the search based approach to selection is not considered by any of these papers. An evolving dataset starting with the initial datasets generated using these approaches can be a good candidate for a search based data selection problem.

Another inspiration for this work is the fact that the public datasets are prone to quality issues and contain noisy data [25]. For example, certain defects might not have been discovered yet, hence, some of the items in the training set may be misleading as non-defective, while with similar kind of measurements defects can exist in the test set. In contrast, while some test instances are not defective, the most similar items in the training set might be labeled as defective. In short, some of the instances in the test set can have similar measurements with the training set, yet different class labels. Please note that mislabeling may not be the only reason for such situations, and they can occur naturally, i.e. the class labels of similar measurements can differ. The acknowledgment of noise in the data and guiding learning to account for that can lead to better predictions, as we propose in this paper.

3.2 Proposed Approach

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

The process starts with splitting the test set into p parts randomly ($p = 5$ in our experiments). Partitioning the test set into smaller chunks plays an important role in the overall procedure. By creating smaller chunks, the process of optimizing and adjusting the dataset is easier as there are less elements to consider and the datasets generated could be better representatives for these smaller chunks than the whole dataset. This procedure however, adds extra complexity to the model and the run-time would increase consequently, but since the goal is to optimize the effectiveness of the method, we leave runtime overheads out of the scope of the paper.

Each part (without the labels) is fed into the (NN)-Filter instance selection method in order to select the most relevant instances from the training set for the purpose of reserving a validation set, on which optimize the search process. Please note that the training set is a combination of all the instances from other projects. The process then randomly creates an initial population containing *popSize* datasets. Each population element is a dataset selected randomly and with replacement from the large pool of training set instances. Each population member is then evaluated on the validation set, which is acquired via the (NN)-Filter in the previous step. Then, for *numGens* generations, a new population is generated and the top elements are selected to survive and move to the next generation. There is an alternative stopping criterion for GIS (described below). These procedures are repeated 30 times to address the randomness introduced by both the dataset selection and genetic operations. Below, the genetic operations and parameters are discussed in more details:

- **Initial Population:** The initial population is generated using the random instance selection process from a large pool of instances containing all elements from other projects than the test project.
- **Chromosome Representation:** Each chromosome contains a number of instances from a list of projects. A chromosome is a dataset sampled from the large training dataset randomly and with replacement.
- **Selection:** The Tournament selection is used as the selection operator of GIS. Since the population size is small in our experiments, tournament size was set to two.
- **Elites:** A proportion of the population is moved to the next generation; those that provide the best fitness values. We transfer two of the top parents to the next generation.
- **Stopping Criteria:** We place two limitations on the number of iterations that the genetic algorithm could progress. The first one is the maximum number of generations allowed. In this case, this number was set to 20. The other stopping criterion is the amount of benefit gained from the population generated. If the difference between the mean fitness of two population is less than $\epsilon = 0.0001$, the genetic algorithm stops.

Algorithm 1 Pseudo code for GIS

```
Set numGens = The number of generations of each genetic optimization run.
Set popSize = The size of the population.
Set DATA = {Ant-1.7, Camel-1.6, ivy-2.0, jedit-4.3, log4j-1.2, lucene-2.4, poi-3.0, prop-6, synapse-1.2, tomcat-6.0, velocity-1.6, xalan-2.7,xerces-1.4}
Set FEATURES = { WMC, DIT, NOC, CBO, RFC, LCOM, LOC }

for TEST in DATA do
    set TRAIN = Instances from all other projects
    tdSize = 0.05 * Number of instances in TRAIN
    for i = 1 to 30 do
        Set TestParts = Split TEST instances into p parts

        for each testPart in TestParts do
            Set vSet = Generate a validation dataset using 5-(NN)-Filter method (using three distance measures).
            Set TrainDataSets = Create popSize dataset from TRAIN with replacement each with tdSize instances

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

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

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

        Calculate Precision, Recall and F-Measure and GMean from the predictions.
    end for
    Report the median of all 30 experiments
end for
```

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

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

- **Crossover:** The generated training datasets used in the population could possibly have large sizes. The time for training a learner with a large dataset and validating it on a medium size validation set increases, if the size of the train and validation datasets increase. To avoid having very large datasets one point crossover was used during the crossover operation. As we have mentioned the chromosomes are a list of instances from the large training set, selected randomly and with replacement. Since the chromosomes possibly contain the repetitions of one item and the mutation operation changes the label of an instance, conflicts might occur in the chromosomes generated from combining the two selected parents. In the case of conflicts, the majority voting is used to select the label of such instances. Algorithm 3 provides the pseudo-code for crossover operation.

Algorithm 2 Mutation

```
Input => DS: a dataset
Output => A dataset with mutated items

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

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

end if
```

3.3 Benchmark Methods

GIS is compared with three other approaches:

- (NN)-Filter (CPDP): In this approach, the most relevant training instances are selected based on a distance measure [30]. In this case, we used 10 nearest neighbours and Euclidean distance. The simplicity of the method and the comprehensive number of studies that have tested the approach are the reasons for choosing this method as a benchmark [23, 24, 30]. Also GIS uses (NN)-Filter to select the validation dataset and a benchmark is required to measure the performance difference between (NN)-Filter and GIS.

Algorithm 3 One point crossover

```
Input => DS1 and DS2
Output => Two new datasets generated from DS1 and DS2

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

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

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

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

- Naive (CPDP): In this approach, the whole training set is fed into the learner and the model is trained with all the training data points. These method has also been tested in many studies and provides a baseline for the comparisons [23, 24, 30]. The approach is easy and at the same time demonstrates that while the availability of large pools of data could be useful, not all the data items are.

- 10-Fold cross validation (WPDP): In this benchmark, we perform stratified cross validation on the test set. Many studies have reported the good or at least better performance of this approach compared with that of cross project methods [30]. Outperforming and improving WPDP is the main goals of many such studies.

While the first two benchmarks (CPDP) are used to answer RQ1, the last benchmark (WPDP) is used to answer RQ2. Each experiment is repeated 30 times to address the randomness introduced by 10 fold cross validation and GIS.

3.4 Datasets and Metrics

We used 13 projects from the PROMISE repository for our experiments. 12 of these projects are open source and one of them is a proprietary project (prop-6). These datasets are collected by Jureczko, Madeyski and Spinellis [12, 13]. The list of the datasets is presented in Table 1 with the corresponding size and defect information. The reason for using these datasets is driven by our goal to account for noise in the data, which is a threat specified by the donors of these datasets. Each dataset contains a number of instances corresponding to the classes in the release. Originally, each instance has 20 static code metrics. Among these 20 metrics, we used seven of them while performing the experiments of this study, as listed in Table 2. Please note that the reason for choosing these specific set of metrics is that they were also used in previous studies [4, 5]. In addition, please note that the selection of the metrics is irrelevant to the topic of this paper as it focuses on the instance selection problem instead of feature selection, and using a reduced set that is tried in other studies allows us to demonstrate the feasibility of our approach as a proof of concept.

Table 1: Datasets used in the study. DP= Defect Prone.

Release	#classes	#DP	DP(%)	#LOC
ant-1.7	745	166	22.3	208,653
camel-1.6	965	188	19.5	113,055
ivy-2.0	352	40	11.4	87,769
jedit-4.3	492	11	02.2	202,363
log4j-1.2	205	189	92.2	38,191
lucene-2.4	340	203	59.7	102,859
poi-3.0	442	281	63.6	129,327
prop-6.0	660	66	10.0	97,570
synapse-1.2	256	86	33.6	53,500
tomcat-6.0	885	77	09.0	300,674
velocity-1.6	229	78	34.1	57,012
xalan-2.7	885	411	46.4	411,737
xerces-1.4	588	437	74.3	141,180

Table 2: List of the metrics used in this study

Variable	Description
CK suite (6)	
WMC	Weighted Methods per Class
DIT	Depth of Inheritance Tree
LCOM	Lack of Cohesion in Methods
RFC	Response for a Class
CBO	Coupling between Object classes
NOC	Number of Children
Lines of Code (1)	
LOC	Lines Of Code

3.5 Performance Measures and Tools

Naïve Bayes (NB) is used as the base learner in all experiments. NB is a member of the probabilistic classifier family that are based on applying Bayes' theorem with strong (naïve) independence assumptions between the features [29]. The good performance of NB has been shown in many studies. Menzies et al. [17, 18] and Lessmann et al.[14] have demonstrated the effectiveness of NB with a set of data mining experiments performed on NASA MDP datasets. Lessmann et al. compared the most common classifiers on the NASA datasets and concluded that there is no significant difference between the performances of top 15 classifiers, one of which is NB [14] .

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

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

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

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

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

Table 3: Median values of the 30 measurements for GIS, (NN)-Filter and Naive approaches.

	Genetic				(NN)-Filter				Naive			
Dataset	Recall	Prec.	F	GMean	Recall	Prec.	F	GMean	Recall	Prec.	F	GMean
ant-1.7	0.946	0.236	0.379	**0.472**	0.295	0.591	0.394	0.417	0.337	0.577	**0.426**	0.441
camel-1.6	0.976	0.193	**0.322**	**0.434**	0.165	0.413	0.236	0.261	0.144	0.450	0.218	0.254
ivy-2.0	0.900	0.135	0.234	0.348	0.450	0.367	0.404	0.407	0.500	0.364	**0.421**	**0.426**
jedit-4.3	0.636	0.019	0.037	0.110	0.455	0.068	**0.118**	**0.175**	0.455	0.057	0.101	0.161
log4j-1.2	0.640	0.925	**0.757**	**0.770**	0.085	1.000	0.156	0.291	0.037	1.000	0.071	0.192
lucene-2.4	0.938	0.605	**0.732**	**0.747**	0.148	0.882	0.253	0.361	0.138	0.903	0.239	0.353
poi-3.0	0.929	0.670	**0.778**	**0.790**	0.107	0.857	0.190	0.303	0.132	0.822	0.227	0.329
prop-6	0.924	0.105	0.189	**0.311**	0.303	0.235	**0.265**	0.267	0.136	0.243	0.175	0.182
synapse-1.2	0.895	0.369	**0.520**	**0.569**	0.256	0.667	0.370	0.413	0.174	0.682	0.278	0.345
tomcat-6.0	0.948	0.089	0.163	0.290	0.597	0.293	**0.393**	**0.418**	0.597	0.279	0.38	0.408
velocity-1.6	0.821	0.367	**0.504**	**0.542**	0.128	0.667	0.215	0.292	0.141	0.688	0.234	0.311
xalan-2.7	0.800	0.992	**0.886**	**0.890**	0.156	1.000	0.270	0.395	0.160	1.000	0.276	0.400
xerces-1.4	0.749	0.805	**0.788**	**0.789**	0.130	0.934	0.229	0.349	0.101	0.978	0.183	0.314
Median	0.896	0.353	**0.496**	**0.542**	0.165	0.667	0.253	0.349	0.144	0.682	0.234	0.329
Mean	0.853	0.424	**0.483**	**0.543**	0.252	0.613	0.269	0.334	0.235	0.619	0.248	0.317

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

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

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

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

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

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

$$F1 - Measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

GMean: While the F-Measure is the harmonic mean of Recall and Precision, the GMean is the geometric mean of the two. GMean is defined as follows:

$$GMean = \sqrt{precision \times recall} \tag{4}$$

In this study, F1-Measure and GMean are selected as the basis of our selection for the best approach when presenting the results. F-Measure and GMean are also used as parts of the fitness function for the genetic algorithm as it was discussed earlier.

All the experiments are conducted using **WEKA**[1] machine Learning tool version 3.6.13. The statistical tests are carried out using the **scipy.stats**[2] library version 0.16.0, **Python**[3] version 3.4.3 and **statistics** library from Python. The plots are generated using the **matplotlib**[4] library version 1.5.1.

4. RESULTS

Table 3 and Table 4 provide the median values of the performance measures from the 30 experiments performed for CPDP and WPDP benchmarks, respectively. In Table 3, the reported results are without variation for (NN)-Filter and naive methods, since there is no randomness involved in their setting. The results of within and cross project predictions are presented separately to show the differences in both within and cross project cases and to answer the corresponding research questions properly. In Table 4, the results of GIS are repeated to make the comparisons easier. In both tables, the last two rows present the median and mean values of all predictions.

To measure the performance of GIS in comparison with the other methods, the Wilcoxon signed rank test is used. Table 5 and Table 6 summarize the results of the statistical tests based on F-Measure and GMean values respectively for each dataset individually and for all 30 runs. The last rows of each table compare all results of GIS with the other approaches. The first column of each approach in these tables is the $p - value$ obtained from the tests and the second column is the Cohen's d value associated with the performance obtained from GIS and the other approach that is subject to comparison. A positive Cohen's d means that GIS yields better results than the compared counterpart. The results of the experiments are also visualized in Figure 2. Note that the thin continuous line in the plots is the median and the thick dashed line represents the mean value of the results.

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

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

Judging from the F-Measure values, we see in Table 3 that the predictions in eight out of 13 cases are improved

[1] http://www.cs.waikato.ac.nz/ml/weka/

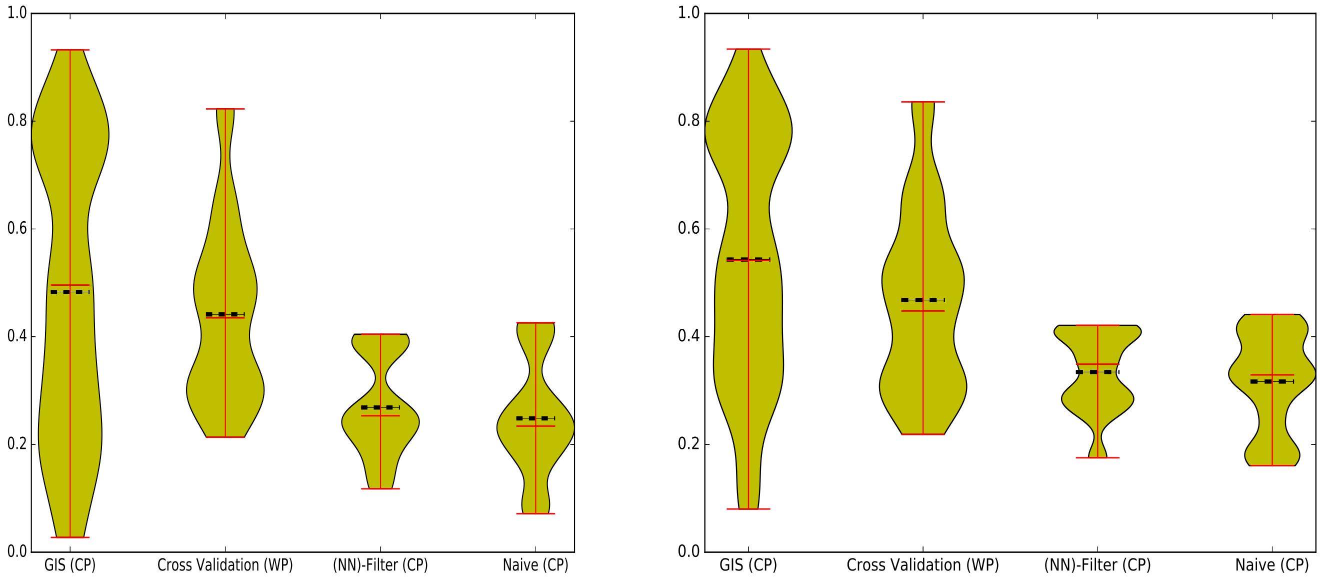
[2] https://www.scipy.org/

[3] http://www.python.org

[4] http://matplotlib.org/

Table 4: Median values of the 30 measurements for GIS and WP 10-Fold Cross Validation approaches.

	Genetic				Cross Validation			
Dataset	Recall	Prec.	F	GMean	Recall	Prec.	F	GMean
ant-1.7	0.946	0.236	0.379	0.472	0.407	0.636	**0.496**	**0.509**
camel-1.6	0.976	0.193	**0.322**	**0.434**	0.187	0.446	0.264	0.289
ivy-2.0	0.900	0.135	0.234	0.348	0.428	0.441	**0.435**	**0.435**
jedit-4.3	0.636	0.019	0.037	0.110	0.273	0.180	**0.217**	**0.222**
log4j-1.2	0.640	0.925	**0.757**	**0.770**	0.408	0.985	0.577	0.634
lucene-2.4	0.938	0.605	**0.732**	**0.747**	0.322	0.882	0.472	0.533
poi-3.0	0.929	0.670	**0.778**	**0.790**	0.233	0.859	0.367	0.448
prop-6	0.924	0.105	0.189	0.311	0.342	0.292	**0.315**	**0.316**
synapse-1.2	0.895	0.369	0.520	**0.569**	0.418	0.684	0.519	0.534
tomcat-6.0	0.948	0.089	0.163	0.290	0.294	0.301	**0.298**	**0.298**
velocity-1.6	0.821	0.367	**0.504**	**0.542**	0.207	0.594	0.307	0.351
xalan-2.7	0.800	0.992	**0.886**	**0.890**	0.695	0.998	0.820	0.833
xerces-1.4	0.749	0.805	**0.788**	**0.789**	0.502	0.921	0.650	0.680
Median	0.896	0.353	**0.496**	**0.542**	0.342	0.636	0.435	0.448
Mean	0.853	0.424	**0.483**	**0.543**	0.363	0.632	0.441	0.468



(a) F-Measure of all predictions from all the datasets

(b) GMean of all predictions from all the datasets

Figure 2: Violin plots of F-Measure and GMean values from all predictions for each approach.

with GIS. The naive approach works better in two cases and the (NN)-Filter approach is better in the remaining three. With GMean, the performance of GIS is even better. The number of test sets that have better predictions is increased to ten out of 13 cases. (NN)-Filter has two and naive CPDP has one better prediction. The overall mean and median values from GIS are higher than that of both benchmark cross project methods for F-Measure and GMean values.

The violin plots of the measurements for F-Measure and GMean values in Figures 2a and 2b provide more insights into the results. GIS has a higher mean, median and max values compared with the CP benchmark methods. Of course, one should note the wider range of the values as well. While GIS performs quite well on most of the datasets, two of them are causing the wide interval. From the results in Table 3, we can see that GIS has difficulties in predicting JEdit dataset and the median F-Measure is only 0.037. At the same time, the performance of GIS on Xalan dataset causes the increase in the max value. Nevertheless, the concentration of the prediction results with GIS is promising. More than half of the predictions are over the maximum values received by the other two CPDP benchmark methods.

Results of the statistical tests and the calculated effect sizes show that GIS is significantly better than the two benchmark CPDP approaches and the effect sizes confirm this conclusion. GIS outperforms (NN)-Filter in terms of F-Measure ($p - value \ll 0.001$, Cohen's $d = 0.697$) and GMean ($p - value \ll 0.001$, Cohen's $d = 0.946$). It also

Table 5: Wilcoxon signed rank test results for F-Measure. GIS (CP) vs. Cross Validation (WP), (NN)-Filter (CP) and Naive (CP). Positives values of Cohen's d show performance improvement with GIS.

	Cross Validation		**(NN)-Filter**		**Naive**	
Dataset	p-value	Cohen's d	p-value	Cohen's d	p-value	Cohen's d
ant-1.7	$\ll 0.001$	-12.415	$\ll 0.001$	-1.434	$\ll 0.001$	-5.074
camel-1.6	$\ll 0.001$	14.246	$\ll 0.001$	19.923	$\ll 0.001$	24.955
ivy-2.0	$\ll 0.001$	-10.646	$\ll 0.001$	-9.163	$\ll 0.001$	-10.068
jedit-4.3	$\ll 0.001$	-24.442	$\ll 0.001$	-11.730	$\ll 0.001$	-9.237
log4j-1.2	$\ll 0.001$	4.079	$\ll 0.001$	13.394	$\ll 0.001$	15.277
lucene-2.4	$\ll 0.001$	23.230	$\ll 0.001$	43.658	$\ll 0.001$	44.924
poi-3.0	$\ll 0.001$	40.823	$\ll 0.001$	59.379	$\ll 0.001$	56.348
prop-6	$\ll 0.001$	-17.911	$\ll 0.001$	-10.872	$\ll 0.001$	2.165
synapse-1.2	0.845	-0.063	$\ll 0.001$	9.205	$\ll 0.001$	14.917
tomcat-6.0	$\ll 0.001$	-22.722	$\ll 0.001$	-42.345	$\ll 0.001$	-40.655
velocity-1.6	$\ll 0.001$	9.087	$\ll 0.001$	13.850	$\ll 0.001$	12.934
xalan-2.7	$\ll 0.001$	1.944	$\ll 0.001$	19.837	$\ll 0.001$	19.605
xerces-1.4	$\ll 0.001$	4.347	$\ll 0.001$	20.306	$\ll 0.001$	22.011
All	$\ll 0.001$	0.227	$\ll 0.001$	0.697	$\ll 0.001$	0.753

Table 6: Wilcoxon signed rank test results for GMean. GIS (CP) vs. Cross Validation (WP), (NN)-Filter (CP) and Naive (CP). Positives values of Cohen's d show performance improvement with GIS.

	Cross Validation		**(NN)-Filter**		**Naive**	
Dataset	P-Value	Cohen's d	P-Value	Cohen's d	P-Value	Cohen's d
ant-1.7	$\ll 0.001$	-3.689	$\ll 0.001$	5.748	$\ll 0.001$	3.313
camel-1.6	$\ll 0.001$	24.553	$\ll 0.001$	28.149	$\ll 0.001$	29.619
ivy-2.0	$\ll 0.001$	-4.196	$\ll 0.001$	-2.808	$\ll 0.001$	-3.778
jedit-4.3	$\ll 0.001$	-5.010	$\ll 0.001$	-2.840	$\ll 0.001$	-2.147
log4j-1.2	$\ll 0.001$	3.588	$\ll 0.001$	12.395	$\ll 0.001$	14.934
lucene-2.4	$\ll 0.001$	16.830	$\ll 0.001$	30.803	$\ll 0.001$	31.452
poi-3.0	$\ll 0.001$	37.629	$\ll 0.001$	45.553	$\ll 0.001$	52.021
prop-6	0.021	-0.513	$\ll 0.001$	4.243	$\ll 0.001$	12.286
synapse-1.2	$\ll 0.001$	2.231	$\ll 0.001$	10.080	$\ll 0.001$	14.436
tomcat-6.0	$\ll 0.001$	-0.855	$\ll 0.001$	-12.208	$\ll 0.001$	-11.154
velocity-1.6	$\ll 0.001$	7.485	$\ll 0.001$	10.252	$\ll 0.001$	9.413
xalan-2.7	$\ll 0.001$	1.901	$\ll 0.001$	17.814	$\ll 0.001$	17.600
xerces-1.4	$\ll 0.001$	3.521	$\ll 0.001$	16.438	$\ll 0.001$	17.772
All	$\ll 0.001$	0.595	$\ll 0.001$	0.946	$\ll 0.001$	0.994

outperforms naive CPDP in terms of F-Measure ($p - value \ll 0.001$, Cohen's $d = 0.753$) and GMean ($p{-}value \ll 0.001$, Cohen's $d = 0.994$). These performance improvements are more visible with GMean, as the effect sizes are larger. We should note that GIS is more focused on recall and has a lower precision while the benchmark approaches focus more on precision and have lower recall values. Even though the fitness function is defined in a way that treats the recall and precision equally, previous studies have shown that the (NN)-Filter (on which the GIS is optimized) focuses on recall more than precision [30]. A fitness function with a focus on precision could optimize the results for achieving values with higher precisions. Of course, this might come with a decrease in the recall as there usually is a trade-off between the two, but careful fitness function selection is one of the key areas to pursue further.

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

Considering Table 4, the performance of GIS in terms of F-Measure is better than that of the within project scenario. Seven out of 13 projects have a better prediction with GIS. Five of the predictions are better in the WP scenario and the performance of both methods on one project is similar (synapse-1.2, $p - value = 0.845$ and Cohen's $d = -0.063$). In terms of GMean, eight of the 13 datasets have better prediction with GIS and five favor WPDP. At the same time, WPDP is better than both benchmark CPDP approaches. As pointed out in the discussions for RQ1, the low precision of GIS on a dataset like JEdit should be noted. Of course a dataset like JEdit seems to be hard to predict since the defect density (0.022%) is very low and the within project scenario's results are not very promising as well (and neither are the benchmark CPDP results).

The mean and median values from GIS are both better than those of WPDP. These differences are in terms of both F-Measure ($p - value \ll 0.001$ and Cohen's $d = 0.227$) and GMean ($p - value \ll 0.001$ and Cohen's $d = 0.595$). Considering the F values, cross validation is worse than GIS, but the effect size is not large whereas the difference between GIS and cross validation is significant in terms of GMean ($p - value \ll 0.001$) and the calculated effect size is medium (Cohen's $d = 0.595$). The shape of the violin plots also support our claim as illustrated in Figure 2b. Especially, they show that the number of predictions that are below the min value of WPDP predictions are much less than the density of predictions with better results from GIS than that of WPDP. Again, we should notice the focus of GIS on recall more than precision while the WP approach is more focused on precision.

5. THREATS TO VALIDITY

During an empirical study, one should be aware of the potential threats to the validity of the obtained results and derived conclusions [33]. The potential threats to the validity identified for this study are assessed in three categories, namely: construct, external and conclusion validity.

5.1 Construct validity

The metrics used in this study are CK and LOC which are different from complexity or other size metrics in some other papers. These metrics have been widely used in previous studies [4, 5, 22]. Even though the static code metrics can achieve good performance [22], the usefulness of this metrics has been widely criticised [17, 30]. The experimental datasets are collected by Jureczko et al. [12, 13], who cautioned that there could be some mistakes in non defective labels as not all the defects had been found (*regex* search through version control commit comments). This may be a potential threat for defect models training and evaluation; on the other hand, this is what GIS is designed to account for.

5.2 External validity

It is difficult to draw general conclusions from empirical studies of software engineering and our results are limited to the analyzed data and context [2]. Even though many researchers have used the same datasets as the basis of their conclusions, there is no assurance about the generalization of conclusions drawn from these projects. Particularly the applicability of the conclusions for commercial, proprietary and closed source software might be different.

5.3 Conclusion validity

Our experiments are repeated 30 times to address the randomness and the results are compared using the Wilcoxon signed rank statistical test. Further, to calculate the magnitude of the difference, Cohen's d for related samples was used as effect size. Another threat is the choice of the evaluation measure. Other researchers might consider different measures to evaluate the methods and as a consequence, some of the observations and conclusions may change. Even though the method works better for the majority of the datasets (compared with both WPDP and CPDP benchmarks), it is not necessarily better for all of them and further investigation is required.

6. CONCLUSIONS

In this study, we presented a novel search based approach to instance selection (GIS). Through an evolutionary process, we tried to converge to an optimal training dataset and at the same time, we considered the potential noise in the labeling of the datasets. We incorporated (NN)-Filter into the model by using it in generating the validation set to optimize the performance of our approach. The proposed method outperforms both within and cross project benchmark methods in terms of F-measure and GMean.

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

Other validation dataset selection techniques using approaches like clustering, distributional characteristics, and tuning the parameters of the genetic model in addition to designing other fitness functions with a focus on different measures are among possible future work to pursue.

7. REFERENCES

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

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

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

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

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

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

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

[8] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Trans. Softw. Eng.*, 38(6):1276–1304, Nov. 2012.

[9] P. He, B. Li, D. Zhang, and Y. Ma. Simplification of training data for cross-project defect prediction. *arXiv preprint arXiv:1405.0773*, 2014.

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

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

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

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

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

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

[16] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data. *West Virginia University, Department of Computer Science*, 2012.

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

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

[19] N. Nagappan and T. Ball. Static analysis tools as early indicators of pre-release defect density. In *Proceedings of the 27th international conference on Software engineering*, pages 580–586. ACM, 2005.

[20] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pages 284–292. IEEE, 2005.

[21] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, pages 452–461. ACM, 2006.

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

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

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

[25] M. J. Shepperd, Q. Song, Z. Sun, and C. Mair. Data quality: Some comments on the NASA software defect datasets. *IEEE Trans. Software Eng.*, 39(9):1208–1215, 2013.

[26] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, May 2005.

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

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

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

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

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

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

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

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

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