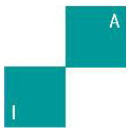
Inteligencia Artificial 19(57) (2016), 114-137
doi: 10.4114/ia.v18i56.1159



INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

An Automated Defect Prediction Framework using Genetic Algorithms: A Validation of Empirical Studies

Juan Murillo-Morera[1], Carlos Castro-Herrera[1] Javier Arroyo[2], Rubén Fuentes-Fernández[2]

Doctoral Program in Computer Science, University of Costa Rica[1]
Department of Software Engineering and Artificial Intelligence, University Complutense of Madrid, Spain[2]
juan.murillomorera@ucr.ac.cr, carlos.castro@ecci.ucr.ac.cr, javier.arroyo@fdi.ucm.es, ruben@fdi.ucm.es

Abstract

Today, it is common for software projects to collect measurement data through development processes. With these data, defect prediction software can try to estimate the defect proneness of a software module, with the objective of assisting and guiding software practitioners. With timely and accurate defect predictions, practitioners can focus their limited testing resources on higher risk areas. This paper reports the results of three empirical studies that uses an automated genetic defect prediction framework. This framework generates and compares different learning schemes (preprocessing + attribute selection + learning algorithms) and selects the best one using a genetic algorithm, with the objective to estimate the defect proneness of a software module. The first empirical study is a performance comparison of our framework with the most important framework of the literature. The second empirical study is a performance and runtime comparison between our framework and an exhaustive framework. The third empirical study is a sensitivity analysis. The last empirical study, is our main contribution in this paper. Performance of the software development defect prediction models (using AUC, Area Under the Curve) was validated using NASA-MDP and PROMISE data sets. Seventeen data sets from NASA-MDP (13) and PROMISE (4) projects were analyzed running a $N \times M$-fold cross-validation. A genetic algorithm was used to select the components of the learning schemes automatically, and to assess and report the results. Our results reported similar performance between frameworks. Our framework reported better runtime than exhaustive framework. Finally, we reported the best configuration according to sensitivity analysis.

Keywords: software quality, fault prediction models, genetic algorithms, learning schemes, learning algorithms, machine learning.

Resumen

Hoy en día, es común medir la complejidad del software a través de sus métricas. Es por medio de las métricas, que se puede estimar la propensión a fallos de un módulo de software, con el objetivo de asistir y orientar a los profesionales de software a realizar sus pruebas. Este trabajo reporta una validación de tres estudios empíricos que utilizan un marco de trabajo genético, el cual de forma automatizada lleva a cabo la predicción de defectos mediante la selección y comparación de diferentes esquemas de aprendizaje (procesamiento + selección de atributos + algoritmos de aprendizaje) con el objetivo de estimar la propensión de defectos de un módulo de software. El primer estudio empírico es una comparación de rendimiento de nuestro marco de trabajo con respecto al marco de trabajo más importante de la literatura. El segundo estudio empírico es una comparación entre el rendimiento y el tiempo de ejecución de nuestro marco de trabajo y un marco de trabajo exhaustivo. Finalmente, el tercer estudio empírico es un análisis de sensibilidad, el cual es nuestra principal contribución en este artículo. El rendimiento de los modelos de predicción de defectos generados (utilizando AUC, área bajo la curva) fueron

ISSN: 1988-3064(on-line)
©IBERAMIA and the authors

validados utilizando conjuntos de datos de las bases históricas: NASA-MDP y PROMISE. Diecisiete conjuntos de datos de la NASA-MDP (13) y PROMISE (4) fueron analizados, ejecutando una validación cruzada $N \times M$. Para la selección del mejor esquema de aprendizaje, se utilizó un algoritmo genético para seleccionar los componentes de los sistemas de aprendizaje de forma automática, y para evaluar y reportar los resultados de los tres estudios empíricos planteados. Nuestros resultados reportaron un rendimiento similar entre los marcos de trabajo. Nuestro marco de trabajo reportó un mejor tiempo de ejecución respecto al marco de trabajo exhaustivo. Por último, se reportó la mejor configuración según el análisis de sensibilidad propuesto.

Palabras Claves: calidad de software, modelos de predicción de fallos, algoritmos genéticos, esquemas de aprendizaje, algoritmos de aprendizaje, aprendizaje máquina.

1 Introduction

With the ubiquity of software and the increasing expectations on quality, the need for fast, reliable software development has seen a rapid growth. As a result of this, research on defect prediction has become a highly important field of software engineering.

Software fault prediction has been an important research topic within software engineering for more than 30 years [43]. Software measurement data collected during the development process comprise valuable information on the status, progress, quality, performance, and evolution of the project. These data are commonly used as input to fault prediction models. Static code attributes as McCabe [27], Halstead [17] and Line of Code can be used to predict defects. These static code attributes are relatively simple to calculate and can be easily automated. The above-cited metrics are **module-based**, where a module is defined as the smallest functionality unit in a program, such as a **function** or **method**. Fault prediction models seek to detect defect prone software modules [46]. The main goal in generating these predictions is to enable software engineers to focus development and testing activities on the most fault-prone parts of their code, thereby improving software quality and making a better use of limited time and resources [16] and [1]. The study and construction of these techniques have been the emphasis of the *fault prediction modeling* research area and also the subject of many previous research projects [12],[22],[23],[33] and [37].

Research on fault prediction is typically broken down into three areas: estimating the number of defects remaining in a software system, discovering defect associations, and classifying software components into defect prone or not [43]. The first approach normally uses statistical methods to estimate the number of defects or the defect density of the code [10],[13] and [30]. The second approach uses association rule mining techniques to reveal software defect associations [44]. Finally, the third approach employs classification techniques to categorize a component as defect prone or not [7],[15],[20],[23],[26],[41],[42] and [46]. This study focuses on the third approach, as it remains a largely unsolved problem.

In recent years, researchers attempting to address the classification problem have begun to use *learning schemes* [43]. At a high level, these learning schemes define a three-step process for building a classifier: first, they perform data pre-processing tasks; then, they select the subset of the available metrics to use (this is referred to as *attribute selection*); and finally, they apply one or more learning algorithms to build the classifier. This paper evaluate an automated genetic defect prediction framework, with the objective to determinate the best possible learning scheme for estimate the defect proneness of a software module. This evaluation is done through a validation of three empirical studies (The first empirical study is a performance comparison of our framework with the most important framework according to the literature. The second empirical study is a performance and runtime comparison between our framework and an exhaustive framework. The third empirical study is a sensitivity analysis). This paper analyzes: 8 Data Preprocessing techniques (DP), 6 Attribute Selectors (AS), and 18 machine Learning Algorithms (LA) representing different kinds of model.

We chose genetic algorithms for three main reasons. According to literature in this field, [24] identified that "there are very few studies that examine the effectiveness of evolutionary algorithms". Representing an open area for future work. She pointed out that "future studies may focus on the predictive accuracy of evolutionary algorithms for software fault prediction". Finally, this research is a combinatory and maximization problem, as both are typical stage settings of genetic algorithms.

Our framework has been compared, in terms of the Song's framework (performance), to an exhaustive framework (performance and speed), varying their genetic configuration (generation, population,

crossover, and mutation).

According to the Goal-Question-Metric (GQM) paradigm [4], the goal of the research can be stated as follows:

Evaluate: An Automated Genetic Defect Prediction Framework
For the purpose of: determinate the best possible learning scheme with the objective to estimate the defect proneness of a software module
With respect to: three empirical studies: The first empirical study is a performance comparison of our framework with the most important framework according to the literature. The second empirical study is a performance and runtime comparison between our framework and an exhaustive framework. The third empirical study is a sensitivity analysis
From the point of view of: the researchers and software engineering practitioners
In the context of: predicting defect proneness in the field of software fault prediction

The content of this article is organized as follow. Section 2 provides background information for the work to be developed. Section 3 presents related work. Section 4 offers research questions related to this study. The proposed framework is explained in Section 5. Genetic configuration is detailed in Section 6. Section 7 describes the experimental design of empirical studies. Section 8 reports results for each empirical study and their statistical analysis. Section 9 addresses threats to validity. Finally, Section 10 drawing the conclusions of the paper and proposes future work.

2 Background

2.1 Metrics

Over thirty years ago, McCabe [27] and Halstead [17] defined a set of static code attributes as metrics that can be used to predict defects. These metrics are **module-based**, where a module is defined as the smallest functionality unit in a program, such as a **function** or **method**. These metrics have been widely used in the field of software quality and defect prediction: [8],[18],[19],[21],[25],[28],[29],[34],[38] and [43].

Halstead metrics seek to measure the complexity of a module based on the number of present operators and operands. Intuition wise, a method with many operations is harder to code and read, and hence more error prone. McCabe metrics, on the other hand, measure complexity by analyzing the intricacy of the pathways of a module. For a more complete explanation of these metrics refer to [28].

2.2 Data sets

There are two main sources of data sets that are widely used by software fault prediction researchers: the PROMISE and the NASA MDP repositories [39] and [40]. These repositories contain data sets from real software projects, where each data set was assessed with regards to metrics and characterized according to attributes. We used the last version of them for this research.

Table 1 shows a summary of data sets (attributes, #fp Mod, %fp Mod). For a complete reference see [43].

2.3 Genetic Algorithms

Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, GAs are by no means random, instead they exploit historical information to direct the search into the region of better performance within the search space. The basic techniques of the GAs are designed to simulate processes in natural systems necessary for evolution, specially those follow the principles first laid down by Charles Darwin of "survival of the fittest". Since in nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones [2] and [47].

Table 1: Data set description

DS	Attributes	Mod	#fp Mod	%fp Mod
CM1	38	505	48	9.50
KC3	38	458	43	9.39
KC4	38	125	61	48.80
MW1	38	403	31	7.69
PC1	38	1107	76	6.87
PC2	38	5589	23	0.41
PC3	38	1563	160	10.24
PC4	38	1458	178	12.21
KC1	21	2107	325	15.42
MC1	39	9466	68	0.72
MC2	40	161	52	32.30
PC5	39	17186	586	3.0
JM1	21	10878	2102	19.32
AR1	29	121	9	7.44
AR3	29	63	8	12.70
AR4	29	107	7	18.69
AR6	29	101	15	14.85

2.4 Learning Schemes

According to [43], a learning scheme consists of three parts: DP, AS and LA. The first part, **DP**, cleans and prepares data. It involves steps such as removing outliers, handling missing values, and transforming numeric values. The second part, **AS**, addresses the selection of the appropriate subset from the attributes available in the data set that will be used as input to the learning algorithm. This is an important task, as it is often the case that there are many attributes collected in a data set that have little or no effect on predicting defects. This is particularly true for data sets that were not created for defect prediction. The last part, **LA**, involves the application of machine learning algorithms that can be used to build the classifier that will predict whether or not a module is defect-prone. For example, some possible learning algorithms are Naive Bayes (NB), Decision Tree (DT), Random Forest (RF), Association Rules (OneR), and SimpleLogistic (SL). For a complete reference see [48].

2.5 Confusion Matrix

A common way of evaluating a classifier is by a confusion matrix (see Figure 1). Columns represent the Predicted classes and rows the Actual classes. True Negatives (TN) are the number of negative examples correctly classified, and True Positives (TP) are the number of positive examples correctly classified. These represent the cases where the classifier determined classes correctly. On the other hand, there are False Negatives (FN) and False Positives (FP), representing the cases where the classifier incorrectly classified something as negative or positive respectively [6] and [9].

	NO (Prediction)	YES (Prediction)
NO (Actual)	True Negative (TN) A	False Positive (FP) B
YES (Actual)	False Negative (FN) C	True Positive (TP) D

Figure 1: Learning Schemes Generator [6]

2.6 Receiver Operating Characteristic (ROC)

The Receiver Operating Characteristic (ROC) curve is a standard technique to summarize classifier performance over a range of trade-offs among true positive and false positive error rates. The Area Under the Curve (AUC) is an accepted performance metric for a ROC curve, and it is widely used [5],[11] and [14]. ROC curves can be thought of as representing the family of best decision boundaries for relative costs of PF and PD. On an ROC curve, the x-axis represents $PF = FP/(FP + TN)$ and the y-axis represents $PD = TP/(TP + FN)$ [43]. The ideal point on a ROC curve would be (0, 100), that is, all positive examples are classified correctly and no negative examples are misclassified as positive [9].

In this paper, AUC is used as the metric to compare performance differences among different learning schemes.

2.7 Cross-validation Test (CV)

Cross-validation (CV) tests are a standard procedure used to evaluate many machine learning algorithms. The general idea behind these tests is to divide training data into a number of partitions, also known as folds. The classifier is evaluated by its classification accuracy on one partition after learning from the remaining ones. This procedure is then repeated until all partitions have been used for evaluation. Some of the most common types are 10-fold, n-fold and bootstrap. The difference among these three types of CV tests lies in the way data are partitioned [15]. We used a N x M-fold cross-validation to calculate the AUC. The AUC final value is the average of N x M executions.

3 Related Work

According to Section 2.2, there are a several works that has used the NASA and PROMISE data sets to estimate the defect proneness of a software module. However, a few of them, have proposed a framework.

In 2007, Menzies et al. [28] published a study in which they compared the performance of two machine learning techniques (Rule Induction and Naive Bayes) to predict software components containing defects. To do this, they used the NASA-MDP repository that, at the time of their research, contained 10 separate data sets. They claimed that "how attributes are used to build predictors is much more important than which particular attributes are used" and "the choice of the learning method is far more important than which subset of available data is used for learning".

In 2011, Song et al. [43] published a study in which they proposed a fault prediction framework based on Menzies's research. They analyzed 12 learning schemes. They argued that, although "how is more important than which", the choice of a learning scheme should depend on the combination of data pre-processing techniques, attribute selection methods, and learning algorithms. Their work confirmed the well-known intrinsic relationship between a learning method and the attribute selection method.

In 2015, following this lead, we published a paper [32] that proposed a genetic fault prediction framework based on Song's architecture. In that work, we selected learning schemes automatically. This automation allows exploring many more possibilities in order to find best learning schemes for **each data set** using a genetic approach. This paper represented the first experimentation and comparison of our framework. Our results reported better performance than Song's framework using ten data sets.

In 2016 [31], we extended this work by conducting a study of how to select the best learning schemes automatically for a specific data set, with a main focus on machine learning algorithms according to their performance (AUC) and using a genetic approach. In this study, we used twelve data sets but without compare our study with others, for example Song's framework using different search spaces (12 and 864). This paper represented our second experimentation and comparison. We compared our framework with an exhaustive framework too. We used ten data sets.

In this paper, we used more data sets (seventeen) than previous works [31] and [32]. We compared our study with others. Song's framework using different search spaces (12 - DP(2)*AS(2)*LA(3) and 864 - DP(8)*AS(6)*LA(18)) and an exhaustive framework with (864 - DP(8)*AS(6)*LA(18)). Finally, we

applied a sensibility analysis, changing the configuration of different genetic operators, with the aim to find the best possible learning scheme per data set. In this study our framework presented results more consolidated and stable.

4 Research Questions

This section lists the main research questions that we set out to answer, according to our three empirical studies: 1) Empirical Study (BaseLine-Song), 2) Empirical Study (BaseLine-Exhaustive) and 3) Empirical Study (Sensitivity Analysis).

- Empirical Study (BaseLine-Song): This empirical study describes a comparison between our framework and with the most important framework according to the literature (Baseline) from the point of view of their performance.

  The questions for this empirical study are:

  - RQ-1.1 Is performance similar between the evaluation(eval) and prediction(pred) phases in relation to other frameworks?
  - RQ-1.2 Is performance similar between Song's prediction phase and our prediction phase with a search space of 12 combinations (same Song's search space)?
  - RQ-1.3 Is the performance similar between Song's prediction phase and our prediction phase with a search space of 864 combinations?
  - RQ-1.4 Which learning schemes are selected between frameworks?

- Empirical Study (BaseLine-Exhaustive): This empirical study describes a comparison between our framework and an exhaustive framework taking into consideration performance and runtime.

  The questions for this empirical study are:

  - RQ-2.1 Is performance similar between an exhaustive framework and our framework?
  - RQ-2.2 Is runtime similar between an exhaustive framework and our framework?
  - RQ-2.3 Which are the data preprocessors, attribute selectors and learning algorithms more frequently selected?

- Empirical Study (Sensitivity Analysis): This empirical study describes a comparison among different genetic configurations (generations, population, crossover, mutation, and replications).

  The questions for this empirical study are:

  - RQ-3.1 Which generation and population configurations reported the best performance?
  - RQ-3.2 Which learning schemes were selected more frequently? (Based on RQ-3.1)
  - RQ-3.3 Which learning schemes reported the best performance considering the mutation levels studied?
  - RQ-3.4 Which learning schemes reported the best performance considering the crossover levels studied?

5 Automated Genetic Framework

5.1 Learning Schemes Generator-Evaluator

To build the prediction models, we mainly followed the framework proposed by Song [43]. The main difference is that our work uses a genetic algorithm to select parts of the learning scheme (preprocessing + attribute selection + learning algorithms). Instead of that, Song's work uses a group of pre-established combinations. In our framework, each element of the learning scheme is part of the chromosome used

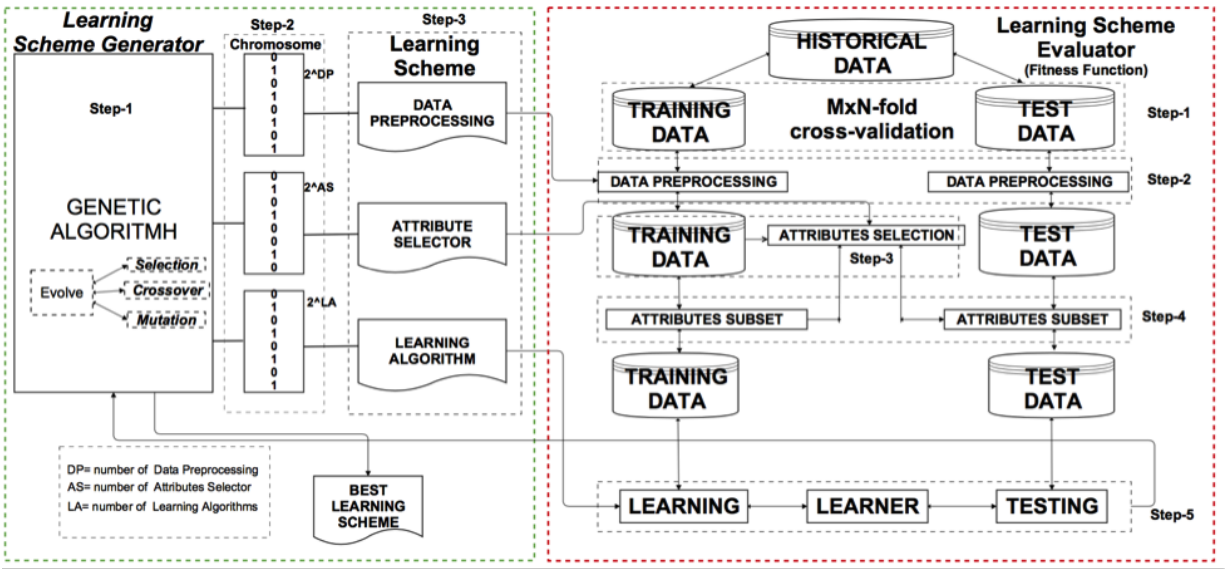


Figure 2: Learning Schemes Generator-Evaluator Adapted from [31] [32]

(for details see Section 6). Our framework consists of two components: 1) Learning Scheme Generator-Evaluator and 2) Defect Prediction. The first one builds the chromosomes and selects the best ones genetically. The second component generates the final predictor that contains the learning scheme selected by the genetic framework. The final predictor uses the learning scheme previously selected and classifies fault-prone modules (true/false).

The Learning Scheme Generator-Evaluator is responsible for generating, evaluating, and selecting the different learning schemes. Selection is done through the elitism technique of the genetic algorithm. Figure 2 shows its components.

The main steps of this process are the following:

1. The population of individuals (chromosomes) is generated and transformed (using as operators selection, crossover and mutation) by the generator component (Fig. 2, **Generator, Step-1**).

2. Each chromosome is represented by a learning scheme (Fig. 2, **Generator, Step-2 and 3**).

3. Historical data were randomized and divided into a training set and a test set. This is done using an $M \times N$-fold cross-validation (Fig. 2, Evaluator **Step-1**).

4. The selected data pre-processing technique is applied to both the training and the test sets (Fig. 2, Evaluator **Step-2**), thus resulting in modified training and test data. This step is represented with the learning method.

5. The chosen attribute selection technique is applied only to the training set (Fig. 2, Evaluator **Step-3**) and the best subset of attributes is chosen.

6. The selected attributes are then extracted for both the training and the test sets (Fig. 2, Evaluator **Step-4** .

7. The Clasiffier is built using the training set, and it is evaluated with the test set. This is performed at (Fig. 2, Evaluator **Step-7**).

Finally, the best learning scheme, the chromosome in our case (with its DP, AS and LA), is selected by the genetic algorithm described in Algorithm 1.

Section 6 provides more details on the different aspects of this frameworks.

5.2 Defect Prediction

The second component of the proposed framework is the defect prediction stage. The main objective of this component is building a predictor for new data, while the main objective of the generator-evaluator of learning schemes (first component) is building a learner and selecting the best one with regard to its AUC.

6 Genetic Configuration

In the field of Artificial Intelligence, a Genetic Algorithm (GA) is a search approach that mimics the biological process of natural selection in order to find a suitable solution in a multidimensional space. Genetic algorithms are, in general, substantially faster than exhaustive search procedures. This section describes the genetic configuration used in this paper: Chromosome, Operators and Fitness Function.

6.1 Chromosome

The chromosome of a GA represents the set of any possible combinations of attributes in the search space. It is commonly represented as a binary chain of 0s and 1s. In this paper, the chromosome consists of three parts: DP, AS, and LA; effectively It constructs a triplet of $< DP, AS, LA >$. We represented DP, AS and LA with binary chains of bits: for DP, we considered 8 possibilities with a chain of 3 bits ($2^3 = 8$); for AS, 6 possible techniques represented by 3 bits ($2^3 = 8$); and for LA, 18 different possibilities requiring 5 bits ($2^5 = 32$). For further details on the considered techniques and their coding, see Section 7.2. With this chromosome representation, the goal of the GA is to find the chromosome maximizing the fitness function.

6.2 Fitness Function

Algorithms 1, 2 and 3 describe the details for the implementation of the fitness function. The final value of AUC is calculated into the prediction phase. (Algorithm 4).

Algorithm 1 shows the $N - PASS$ phase, where the fitness score of a chromosome is calculated as the average of 10 runs. Before this algorithm, the whole data set is split randomly into histData (90% used for training and testing) and newData (10%), and this is repeated for each $PASS$.

The coding process generates a chain of bits (genotype). On the other hand, the decoding process (phenotype) is calculated with the fitness function into the evaluation function (Algorithm 1, line 5). We selected the DP, AS, and LA conforming to the phenotype of each part of the chromosome, represented by the classifier used in (Algorithm 3, line 5). The final phenotype is represented by the AUC (performance) of a specific learning scheme (chromosome), calculated into Algorithm 4.

Algorithm 1 Fitness Function based on [31]

Require: $individual : Chromosome$
Ensure: $AUC \leftarrow Real$
1: $LS \leftarrow individual.getlS()$
2: $i, AUC \leftarrow 0$
3: $NPASS \leftarrow 10$
4: **while** $i < NPASS$ **do**
5: $\quad AUC \leftarrow AUC + Evaluation(histData[i], LS)$
6: $\quad i \leftarrow i + 1$
7: **end while**
8: **return** $AUC/NPASS$

Algorithm 2 shows the evaluation phase. This algorithm performs a N x M-fold cross-validation, where multiple rounds are executed with different partitions in order to reduce variability. Validation results are averaged over the rounds.

Algorithm 3 shows how to calculate the classifier. The classifier uses the learning scheme selected.

Algorithm 2 Evaluation based on [31]

Require: $histData : Instances$, $LS : learningScheme$
Ensure: $EvalAUC \leftarrow Real$
1: $trainDS, testDS : instances$
2: $Cls : Classifier$
3: $i, j, EvalAUC \leftarrow 0$
4: $M, N, folds \leftarrow 10$
5: $histData \leftarrow histData.random()$
6: **while** $i < N$ **do**
7: **while** $j < M$ **do**
8: $trainDS \leftarrow histData.getAllFoldsExcept(j)$
9: $testDS \leftarrow histData.getFold(j)$
10: $Cls \leftarrow Learning(trainDS, LS)$
11: $EvalAUC \leftarrow EvalAUC + evaluateModel(Cls, testDS)$
12: $j \leftarrow j + 1$
13: **end while**
14: $i \leftarrow i + 1$
15: **end while**
16: **return** $EvalAUC/(N * M)$

Algorithm 3 *Learning*

Require: $(trainDS[] : Instances$, $LS : learningScheme$
Ensure: $dP : Instances$, $retArr : Integer[]$, $attsel : Metrics$, $Cls : Classifier$
1: **if** $TypeOfPreprocessing <> NONE$ **then**
2: $trainDS[i] \leftarrow ApplyPreProcessing(trainDS[i])$
3: **end if**
4: $attsel.SelectAttributes(trainDS)$
5: $retAtt \leftarrow attsel.selectedAttributes()$
6: $dP \leftarrow Filter(trainDS, retAtt)$
7: $Cls \leftarrow lA.buildClassifier(dP, LS)$
8: **return** Cls

Algorithm 4 Prediction based on [31]

Require: $historicalData[], newData[] : Instances$, $LS : learningScheme$
Ensure: $PredAUC$
1: $i \leftarrow 0$
2: $NPASS \leftarrow 10$
3: **while** $i < NPASS$ **do**
4: **if** $TypeOfPreprocessing <> NONE$ **then**
5: $newData[i] \leftarrow ApplyPreProcessing(newData[i])$
6: **end if**
7: $PredAUC \leftarrow PredUAC + prediction(historical[i], newData[i], LS)$
8: **end while**
9: **return** $PredAUC/NPASS$

Finally, Algorithm 4, illustrates the prediction phase. This algorithm generates the final value of AUC using the unseen newData and the best learning scheme (LS) calculated in the evaluation phase.

6.3 Operators

The operators of selection, reproduction, crossover, and mutation used here were configured using the default values provided by WEKA in its genetic search [48]. These are: population size= 20, crossover probability = 0.6, 0.7 and 0.9, mutation probability = 0.01, 0.033 and 0.1. and elitism = true.

7 Empirical Study

7.1 Data Sets

To conduct these experiments, we used the following data sets: CM1, KC3, KC4, MW1, PC1, PC2, PC3, PC4, KC1, MC1, MC2, PC5, JM1, AR1, AR3, AR4 and AR6 from the NASA-MDP. For further details, see [43].

7.2 Learning Schemes

As stated in Section 2.4, learning schemes consist of three parts: Data Pre-processing, Attribute Selector, and Machine Learning Algorithms (refer to learning algorithm). This section presents a detail of the different techniques used for each part. In all, we tested 864 different learning schemes [31] and [48].

- *Data Pre-processing (DP)*: None, Log, BoxCox λ=-2 (BC-2), λ=-1 (BC-1), λ=-0.5 (BC-0.5), λ=0.5 (BC0.5), λ=1 (BC1), and λ=2 (BC2).
- *Attribute Selector (AS)*: Backward Elimination(BE), Forward Selection(FS), BestFirst(BF), LinearForwardSelection(LFS), RankSearch(RS) and GeneticSearch(GS).
- *Learning Algorithm (LA)*: NaiveBayes (NB), BayesNet (BN), BayesianLogisticRegression (BLR), NaiveBayesSimple (NBS), Logistic (LOG), SimpleLogistic (SL), MultilayerPerceptron (MP), Bagging (BAG), Dagging (DAG), LogitBoost (LGB), MIBoost (MIB), OneR (OneR), ZeroR (ZeroR), J48 (J48), RandomForest (RF), REPTree(REPT), NBTree (NBT) and RandomTree (RT).

7.3 General experimental design

The experimental process is described as follows:

1. In our experiment, we used $n = 17$ data sets (see Section 7.1).

   The general experimental process had the following characteristics:

   (a) For the implementation, we used JGAP-API [36].

   (b) The search space presented a total of 864 combinations (DP(8)*AS(6)*LA(18)) (see Section 7.2).

   (c) We generated the populations and generations randomly, using the standard configuration of WEKA's geneticSearch [48]. Each genetic individual was represented by a learning scheme.

   (d) In the generation-evaluation phase we used the following configuration: mutation (0.01), crossover (0.6). We used tournament as operator of selection, with tournamentk = 2 and tournamentp = 0.5. Finally, we set elitism to true.

   (e) In the generation-evaluation phase, we applied a strategy for the selection of attributes called Wrapper [48]. It was used with the objective of selecting the attributes for each subset using an internal cross-validation. Wrappers generally provide better results than filters, but they are more computationally intensive [43].

(f) We set $N - PASS = 10$, and calculated the AUC average after $N - PASS$ runs. For each $PASS$, we selected 90% of the data as historical at random.

(g) An $N = 10$ x $M = 10$-fold cross-validation was used to evaluate each learning scheme. Modules were selected at random. Furthermore, the evaluation metrics AUC, Recall and Precision and their average were calculated after N x M-fold cross-validation.

(h) The fitness function of each genetic individual was executed in 1000 holdout experiments, ($N - PASS = 10$) and $N = 10$ x $M = 10$-fold cross-validation. The mean of the 1000 AUC measures was reported as the evaluation performance per genetic individual. The historicalData (90%) was preprocessed considering preprocessing techniques. Then, the predictor was used to predict defects with the newData (10%), which was preprocessed the same way as the historical data.

(i) Finally, steps (a) to (g) were executed 10 times in order to study the steadiness of our framework.

7.4 Specific settings for the empirical studies

First we applied the steps of the general experimental design. For details see Section 7.3. After applying the general experimental design, we executed the following specific steps:

1. A version of Song Framework was implemented, following the steps of each algorithm according to [43].
2. We executed both frameworks (Genetic and Song) using the same 12 combinations mentioned in [43]. Thus, the combinations used were DP (None and Log), AS (Backward Elimination and Forward Selection) and LA (Naive Bayes, J48 and OneR). The main objective of this experiment is to compare our proposal to the same search space used by Song.
3. We executed the Genetic Framework using 864 combinations (see Section 7.2), and we compared it to the 12 combinations mentioned in [43]. The main objective of this experiment is to find other combinations of learning schemes with the best performance.
4. A version of the exhaustive framework was implemented, following the same steps of each algorithm based on [43], but, this time, with the difference that 864 combinations were executed. For details see Section 7.2.
5. We executed both frameworks (Genetic and Exhaustive) using the same 864 combinations mentioned in this paper.
6. Our selection of the population, generation, mutation and crossover levels is based on [3].
7. We executed the genetic framework with three levels of Generation and Population. The Generation (10, 20, 40). and the Population (10, 20, 40).
8. We executed the genetic framework with three levels of Mutation (0.01, 0.033, 0.1) and three levels of Crossover (0.6, 0.7, 0.9).

8 RESULTS AND ANALYSIS

We executed three empirical studies. The first empirical study is a performance comparison of our framework with the most important framework according to the literature. The second empirical study is a performance and runtime comparison between our framework and an exhaustive framework. The third empirical study is a sensitivity analysis. For each empirical study we calculated their eval (evaluation) and pred (prediction) phase. The result of the evaluation phase is calculated in 5.1. Finally, the result of the prediction phase is calculated in 5.2.

We applied a non-parametric test called Wilcoxon rank test. This test substitute for the t-test for paired samples. The desirable minimal number of paired samples is 10 and it is expected that the population would have a median, be continuous and symmetrical. The differences between the variates

are tabulated and ranked; the largest receives the highest rank. In the case of ties, each should be assigned to a shared rank. The smaller group of signed-rank values is then summed as the T value. This T value is compared with figures in a statistical table. If the value obtained is smaller than that in the body of the table under probability and on the line corresponding to the number of pairs tested, then the null hypothesis is rejected and the conclusion is justified that the two samples are different [35].

8.1 Empirical Study (BaseLine-Song)

This empirical study describes a comparison between our framework and others frameworks (Baseline) from the point of view of their performance. We compared our framework with 12 combinations and 864 combinations. Table 2 shows the results of both frameworks using 12 combinations [43].

Table 2: Genetic Framework Performance (with 12 combinations)

DS	Genetic			Song		
	Eval	Pred	LS	Eval	Pred	LS
CM1	0.75	0.77	NONE+BE+NB	0.78	0.78	NONE+BE+NB
KC3	0.81	0.80	NONE+FS+NB	0.83	0.83	LOG+BE+NB
KC4	0.79	0.80	NONE+BE+NB	0.8	0.81	LOG+BE+NB
MW1	0.70	0.75	LOG+BE+NB	0.78	0.77	NONE+FS+NB
PC1	0.75	0.75	LOG+BE+NB	0.78	0.79	LOG+BE+NB
PC2	0.77	0.79	LOG+FS+NB	0.87	0.88	LOG+FS+NB
PC3	0.80	0.81	LOG+FS+NB	0.81	0.81	LOG+FS+NB
PC4	0.86	0.90	LOG+FS+NB	0.90	0.90	LOG+FS+NB
KC1	0.77	0.89	LOG+BE+NB	0.79	0.80	NONE+FS+NB
MC1	0.88	0.80	LOG+BE+NB	0.94	0.94	LOG+FS+NB
MC2	0.73	0.71	LOG+BE+J48	0.71	0.71	NONE+BE+NB
PC5	0.95	0.96	LOG+BE+NB	0.96	0.96	LOG+BE+NB
JM1	0.73	0.71	LOG+FS+J48	0.73	0.71	LOG+FS+J48
AR1	0.61	0.75	LOG+BE+J48	0.63	0.78	LOG+FS+NB
AR3	0.72	0.74	LOG+FS+NB	0.73	0.75	NONE+BE+NB
AR4	0.70	0.76	LOG+BE+NB	0.78	0.79	NONE+BE+NB
AR6	0.73	0.73	LOG+BE+NB	0.64	0.71	NONE+FS+NB

- RQ-1.1 Is performance similar between the evaluation(eval) and prediction(pred) phases in relation to other frameworks?

  Table 2 presents a summary of the performance of our genetic framework for 12 combinations. Our hypotheses is that performance is similar between the evaluation and prediction phases in relation to other frameworks. Our result was $p_{value} = 0.08648 > \alpha = 0.05$. This means that we did not find a statistically significant difference between the evaluation and prediction phases for 12 combinations.

  Table 2 shows the results of all data sets studied. The results reported by the genetic framework are very similar and stable. For example, the data sets with more difference according to evaluation in the evaluation (eval) and prediction (pred) phases were AR1 with a difference of 0.14 and KC1 with 0.12, representing 5.88% each. The rest of the data sets reported differences between phases of 0.00 to 0.08. The group of data that did not report any differences when compared to Song framework was PC1 and AR6 representing 11.76%. On the other hand, the group that reported a difference of 0.01 was KC3, KC4, PC3 and PC5 representing 23.52% of the total. The group with a performance of 0.02 was CM1, PC2, MC2, JM1 and AR3, representing 29.41% of the total. The rest of the data sets (MW1, PC4, MC1 and AR4) reported a performance between 0.04 and 0.08, representing 23.52% of the total. All the data sets reported better performance in the prediction phase, except for KC3 and MC1.

- RQ-1.2 Is performance similar between Song's prediction phase and our prediction phase with a search space of 12 combinations (same Song's search space)?

Table 3: Genetic Framework Performance (with 864 combinations)

DS	Genetic			Song		
	Eval	Pred	LS	Eval	Pred	LS
CM1	0.7911	0.8050	(BC-0.5)+LFS+LOG	0.7821	0.7856	NONE+BE+NB
KC3	0.7859	0.8139	(BC-0.5)+LFS+NBS	0.8323	0.8311	LOG+BE+NB
KC4	0.9066	0.9051	(BC-0.5)+LFS+LOG	0.8543	0.8163	LOG+BE+NB
MW1	0.8786	0.8886	(BC1)+LFS+NB	0.7843	0.7781	NONE+FS+NB
PC1	0.8384	0.8220	NONE+BE+BAG	0.7863	0.7983	LOG+BE+NB
PC2	0.8745	0.9084	(BC-2)+FS+LOG	0.8734	0.8882	LOG+FS+NB
PC3	0.8223	0.8236	(BC-0.5)+LFS+MP	0.8161	0.8184	LOG+FS+NB
PC4	0.9264	0.9236	(BC-1)+FS+LGB	0.9055	0.9082	LOG+FS+NB
KC1	0.8060	0.7082	(BC1)+LFS+BAG	0.7929	0.8029	NONE+FS+NB
MC1	0.9820	0.9808	NONE+LFS+MP	0.9489	0.9402	LOG+FS+NB
MC2	0.8145	0.8155	(BC1)+LFS+NBS	0.7176	0.7132	NONE+BE+NB
PC5	0.9830	0.9840	NONE+LFS+SL	0.9636	0.9682	LOG+BE+NB
JM1	0.7342	0.7332	(BC-1)+LFS+BAG	0.7165	0.7194	LOG+FS+J48
AR1	0.7353	0.7318	(BC-1)+BE+MP	0.6385	0.7894	LOG+FS+NB
AR3	0.7589	0.7524	(BC-2)+GS+MP	0.7340	0.7502	NONE+BE+NB
AR4	0.7969	0.8083	NONE+LFS+NBS	0.7840	0.7902	NONE+BE+NB
AR6	0.7124	0.7260	(BC-2)+BE+MP	0.6474	0.7139	NONE+FS+NB

Table 2 presents a summary of the performance of our genetic framework for 12 combinations. Our hypotheses is that performance is similar between the prediction phases in relation to other frameworks. Our result was $p_{value} = 0.06453 > \alpha = 0.05$. This means that we did not find a statistically significant difference between prediction phases with a search space of 12 combinations.

Table 2 shows the results of the all data sets evaluated. The results between the prediction phases of both frameworks reported similar results. The data sets PC3, PC4, MC2, PC5 and JM1 did not present difference, representing 29.41% of the total. The rest of the data sets (64.71% of the total) reported values between 0.01 (CM1 and KC4) - 0.09 (KC1). The exception was MC1 with a difference of 0.14, representing 5.88% of the total.

- RQ-1.3 Is the performance similar between Song's prediction phase and our prediction phase with a search space of 864 combinations?

  Table 3 presents a summary of the performance of our genetic framework for 864 combinations. Our hypotheses is that performance is similar between the prediction phases in relation to other frameworks when the search space increased from 12 to 864 combinations. The ($p_{value} = 0.007995 < \alpha = 0.05$). This means that we did find a statistically significant difference between the prediction phases when the search space increased from 12 to 864 combinations.

  According to table 3 Our framework presented better prediction results than Song's. This means that it is important to explore more learning algorithms (data preprocessing, attribute selectors and learning algorithms). According to performance, we have three groups of data sets. The first group of data sets reported better performance than Song'framework. The results were CM1, KC4, MW1, PC1-PC4, MC1, MC2, PC5, JM1, AR4 and AR6. The second group of data sets reported similar performance than Song' framework. This group was KC1 and AR3. Finally, the third group of data sets (AR1 and KC3) reported worst performance than Song's framework. The first group represents the 76.47% of the total, with a performance lower than Song' framework. The second group represents the 11.76% of the total, with a performance similar than Song' framework. Finally, the third group represents the 11.76% of the total with a performance lower than Song' framework.

- RQ-1.4 Which learning schemes are selected between frameworks?

  – For the 12 combinations

Table 2 shows the Genetic and Song learning schemes with the best performance per data set. According to these results, the more predominant learning schemes were LOG+BE+NB representing (47.05%) of the total. The second more predominant learning scheme was LOG+BE+NB, representing (17.64%).

– For the 864 combinations

Table 3 shows the Genetic and Song learning schemes with the best performance per data set. According to the results, most of the learning schemes, representing (88.23%), did not report a predominant learning scheme among data sets, except for (BC-0.5)+LFS+LOG for the CM1 and KC4 data sets, representing (11.77%). This result is validated by literature [28], [43]. However, the most predominant data pre-processing were: BoxCox with a representation of (76.47%) and NONE (23.53%). The most predominant attribute selector were LFS (64.70%) and BE (17.64%). Finally, the main learning algorithms were MP (29.41%) and BAG, NBS and LOG (17.64%) each.

8.2 Empirical Study (BaseLine-Exhaustive)

This empirical study describes a comparison between our framework and an exhaustive framework (Baseline) from the point of view of their performance. The main objective is validate the performance results between our framework and an exhaustive framework (contain all the possible solutions) and analyze differences.

- RQ-2.1 Is performance similar between an exhaustive framework and our framework?

  Table 4 presents a summary of the performance of our genetic framework for 864 combinations. Our hypotheses is that performance is similar between an exhaustive framework and our framework. Our result was $p_{value} = 0.06334 > \alpha = 0.05$. This means that we did not find a statistically significant difference between frameworks using 864 combinations.

  Table 4 shows the results of the implementation of both analyzed frameworks. According to the prediction phase, the performance of our framework is very similar to that of the exhaustive framework. A first group of data sets: CM1, KC4, MW1, PC3, PC4, KC1, MC1, MC2, PC5, AR1, AR3, AR4 and AR6, reported the same prediction values.

  This group represents 76.47%. On the other hand, a second group of data set: KC3, PC1, PC2 and JM1, reported in our framework prediction values lower than those of the exhaustive framework, representing 23.53%. As exceptions, KC3 an JM1 presented a AUC difference of 0.01 and 0.02 respectively, when compared with the exhaustive framework.

- RQ-2.2 Is runtime similar between an exhaustive framework and our framework?

  Table 5 presents a summary of the runtime of our genetic framework for 864 combinations. Our hypotheses is that runtime is similar between an exhaustive framework and our framework. Our result was $p_{value} = 0.001282 < \alpha = 0.05$. This means that we did find a statistically significant difference between runtimes, where our genetic framework reports a better runtime than the exhaustive framework.

  According to the results, there is a clear difference between the runtimes of the genetic framework when compared to the exhaustive framework. In all cases, our genetic framework reported better runtimes than the exhaustive framework. For example, as reported by Table 5, the data set that reported the lowest runtime difference between frameworks were AR1, with 26 minutes, and MC2 with 37 minutes. On the other hand, the data sets that reported highest runtime difference were PC5 with 16904 minutes, MC1 with 6142 minutes, and JM1, with 5643 minutes. This also clearly indicate that the size of data set is an important fact. This an important reason to use genetic algorithms because according to the literature, the GAs have lower runtime than the exhaustive approaches.

- RQ-2.3 Which are the data preprocessors, attribute selectors and learning algorithms more frequently selected?

Table 4 shows the main learning schemes per data sets, according to their performance. For the genetic framework, the main data preprocessors were (BC-0.5) with 23.53%, (BC1),(BC-1) and (BC-2) with 17.64% each. The main attribute selectors were LFS with 64.70% and BE with 17.64%. Finally, the most selected learning schemes were MP with 29.41%, LOG, BAG and NBS with 17.64% each. Otherwise, for the exhaustive framework, the main data preprocessors were (BC-2) with 23.53%, (BC-0.5), LOG and (BC-1) with 17.64% each. The main attribute selectors were LFS with 47.05% and GS with 29.41%. Finally, the most selected learning schemes were NB with 23.52%, NBS, MP and BAG with 17.64% each.

Table 4: Performance between Genetic and Exhaustive Framework(Prediction)

DS	Genetic			Exhaustive	
	Eval	Pred	LS	Pred	LS
CM1	0.7911	0.8050	(BC-0.5)+LFS+LOG	0.8089	(BC-0.5)+LFS+NB
KC3	0.7859	0.8139	(BC-0.5)+LFS+NBS	0.8252	(BC-0.5)+GS+NBS
KC4	0.9066	0.9051	(BC-0.5)+LFS+LOG	0.9056	(BC-2)+LFS+LOG
MW1	0.8786	0.8886	(BC1)+LFS+NB	0.8863	NONE+LFS+NB
PC1	0.8384	0.8220	NONE+BE+BAG	0.8372	LOG+GS+LGB
PC2	0.8745	0.9084	(BC-2)+FS+LOG	0.9156	(BC-2)+BE+LOG
PC3	0.8223	0.8236	(BC-0.5)+LFS+MP	0.8242	(BC-0.5)+LFS+NB
PC4	0.9264	0.9236	(BC-1)+FS+LGB	0.9295	(BC-1)+BE+LGB
KC1	0.8060	0.7082	(BC1)+LFS+BAG	0.8081	(BC-1)+LFS+BAG
MC1	0.9820	0.9808	NONE+LFS+MP	0.9883	NONE+BE+MP
MC2	0.8145	0.8155	(BC1)+LFS+NBS	0.8181	LOG+LFS+NBS
PC5	0.9830	0.9840	NONE+LFS+SL	0.9864	NONE+LFS+NB
JM1	0.7342	0.7332	(BC-1)+LFS+BAG	0.7543	LOG+LFS+BAG
AR1	0.7353	0.7318	(BC-1)+BE+MP	0.7363	(BC-1)+GS+MP
AR3	0.7589	0.7524	(BC-2)+GS+MP	0.7578	(BC-2)+GS+MLP
AR4	0.7969	0.8083	NONE+LFS+NBS	0.8098	NONE+BF+NBS
AR6	0.7124	0.7260	(BC-2)+BE+MP	0.7298	(BC-2)+GS+MP

8.3 Empirical Study (Sensitivity Analysis)

This empirical study describes a comparison among different genetic configurations (generations, population, crossover and mutation). This corresponds to the sensitivity analysis of our framework. This analysis allows verifying the values of generations, population, mutation and crossover [45].

- RQ-3.1 Which generation and population configurations reported the best performance?

  For the population and generation, we assessed two hypotheses. 1) To evaluate the performance of the prediction phase of our genetic framework with a population and generation (10x10 and 20x20). 2) To evaluate the performance of the prediction phase of our genetic framework with a population and generation (20x20 and 40x40). Table 6 presents a summary of performance in the prediction phase of our genetic framework. For the first hypothesis, $p_{value} = 0.0005035 < \alpha = 0.05$. This means that we did find a statistically significant difference between configurations (10x10 and 20x20) in the genetic framework. For the second hypothesis, $p_{value} = 0.2435 > \alpha = 0.05$. This means that we did not find a statistically significant difference between configurations (20x20 and 40x40).

  Table 6 shows the results of three levels of generations and populations (10x10, 20x20, 40x40). According to performance, the best configuration was (40x40) for all data sets. However, the runtime of this configuration presented a statistically significant difference compared to the exhaustive framework runtime ($p_{value} = 0.04437 < \alpha = 0.05$). The mean performance between data sets was 0.805. This is the main reason why we selected the configuration (20x20) for the analysis of previous empirical studies.

Table 5: Runtime (minutes) between Genetic and Exhaustive Framework

DS	Modules	Genetic	Exhaustive	Difference
CM1	505	64	370	306
KC3	458	37	136	99
KC4	125	35	140	105
MW1	403	100	213	113
PC1	1107	125	595	470
PC2	5589	85	463	378
PC3	1563	75	1672	1597
PC4	1458	183	2150	1967
KC1	2107	68	926	858
MC1	9466	3155	9297	6142
MC2	161	48	85	37
PC5	17186	5664	22568	16904
JM1	10878	2821	8464	5643
AR1	121	40	66	26
AR3	63	21	60	39
AR4	107	31	70	39
AR6	101	22	62	42

- RQ-3.2 Which learning schemes are selected most frequently? (Based on RQ-3.1)

  Table 6 shows the main learning schemes per data set for the configuration (20x20). The most important data preprocessing techniques were (BC0.5) with 41.17% and (BC-1), (BC-2) with 17.64% each. The most important attribute selectors were LFS with 47.05% and FS with 23.52%. Finally, the most important learning algorithms were MP with 29.41% and LOG, BAG and NBS with 17.64% each. An important pattern found it for this configuration was (BC0.5) + LFS + MP.

  Table 6 shows the main learning schemes per data set for the configuration (40x40). The most important data preprocessing techniques were (BC0.5) with 35.29% and (BC-1) with 23.52%. The most important attribute selectors were FS with 35.29% and RS with 29.41%. Finally, the most important learning algorithms were MP with 35.29% and LOG with 23.52%. An important pattern found it for this configuration was (BC0.5) + FS + MP.

- RQ-3.3 Which learning schemes reported the best performance considering the mutation levels studied?

  For the mutation operator, we assessed two hypotheses. 1) To evaluate the performance of the prediction phases of our genetic framework with a mutation (0.01 and 0.033). 2) To evaluate the performance of the prediction phases of our genetic framework with a mutation (0.01 and 0.1). Table 7 presents a summary of performance in the prediction phases of our genetic framework. For the first hypothesis, $p_{value} = 0.03479 < \alpha = 0.05$. This means that we did find a statistically significant difference between the configurations (0.01 and 0.033) of the genetic framework. For the second hypothesis, $p_{value} = 0.01347 < \alpha = 0.05$. This means that we did find a statistically significant difference between configurations (0.01 and 0.033).

  Table 7 shows the results of different leves of mutation (0.01, 0.033, 0.1). The mutation rate with the best performance was 0.01 with 76.47%, 0.033 and 0.1 with 17.64% each. According to Table 7, the best performance results per data set are indicated in bold.

- RQ-3.4 Which learning schemes reported the best performance considering the levels of crossover studied?

  For the crossover operator, we assessed two hypotheses. 1) To evaluate the performance of the prediction phases of our genetic framework with a crossover (0.6 and 0.7). 2) To evaluate the performance of the prediction phases of our genetic framework with a crossover (0.6 and 0.9). Table 8 presents a summary of performance in the prediction phases of our genetic framework. For

the first hypothesis, $p_{value} = 0.02322 < \alpha = 0.05$. This means that we did find a statistically significant difference between the configurations (0.6 and 0.7) of the genetic framework. For the second hypothesis, $p_{value} = 0.02452 < \alpha = 0.05$. This means that we did find a statistically significant difference between configurations (0.6 and 0.9).

Table 8 shows the results of different leves of crossover (0.6, 0.7, 0.9). The crossover rate with the best performance was 0.6 with 58.82%, 0.9 with 41.17% and 0.7 with 11.76%. According to Table 8, the best performance results per data set are indicated in bold.

9 THREATS TO VALIDITY

Internal validity: Statistical results show that our approach is steady and robust. Our hypothesis was successfully proven. However, more experiments are required to validate other configurations. For example, more experimentation using different genetic operators implementations. For this study, we used $N - PASS = 10$, $HistoricalDS = 90\%$, $NewDS = 10\%$, $CrossValidation = (NxM = 10x10)$ and $Wrapper = (NxM = 10x10)$. More different configurations are needed varying the parameters of the configurations proposed.

External validity: The results of this study only consider public data sets (NASA-MDP and PROMISE). Thus, more experimentation is needed with real life projects presenting more missing values, imbalanced data, and outliers, among others.

Construction validity: Wrappers generally provide better results than filters, but they are more computationally intensive. In our proposed framework, we use the wrapper evaluation method.

Table 6: Population and Generation

	Genetic									
	10x10			20x20			40x40			
DS	Eval	Pred	LS	Eval	Pred	LS	Eval	Pred	LS	
CM1	0.7336	0.6661	NONE+BE+NB	0.7911	0.8050	(BC0.5)+LFS+LOG	0.7923	**0.8098**	(BC0.5)+RS+MP	
KC3	0.6866	0.5850	NONE+BE+NBS	0.7859	0.8139	(BC0.5)+LFS+NBS	0.7145	**0.8532**	NONE+FS+NBS	
KC4	0.8720	0.8823	(BC0.5)+LFS+LOG	0.9051	0.9066	(BC0.5)+LFS+LOG	0.9134	**0.9198**	(BC0.5)+RS+LOG	
MW1	0.7102	0.6529	(BC-2)+RS+LOG	0.8786	0.8886	(BC0.5)+LFS+NB	0.9011	**0.9023**	(BC-0.5)+LFS+LOG	
PC1	0.7888	0.7996	LOG+FS+LOG	0.8384	0.8220	NONE+BE+BAG	0.8367	**0.8389**	NONE+FS+BAG	
PC2	0.6604	0.7231	(BC-0.5)+LFS+LOG	0.8745	0.9084	(BC-2)+FS+LOG	0.8806	**0.9093**	LOG+FS+LOG	
PC3	0.8191	0.7997	(BC-0.5)+LFS+MP	0.8223	0.8236	(BC0.5)+LFS+MP	0.8312	**0.8373**	(BC-1)+LFS+MP	
PC4	0.8093	0.8977	(BC0.5)+LFS+BAG	0.9264	0.9236	(BC-1)+FS+LGB	0.9282	**0.9273**	(BC0.5)+LFS+DAG	
KC1	0.6423	0.7946	(BC0.5)+LFS+BAG	0.8060	0.7082	(BC0.5)+FS+BAG	0.8133	**0.8154**	(BC0.5)+RS+MP	
MC1	0.9802	0.9804	LOG+GS+MP	0.9820	0.9808	LOG+GS+MP	0.9823	**0.9845**	LOG+GS+MP	
MC2	0.8009	0.7970	(BC-2)+FS+LOG	0.8145	0.8155	(BC0.5)+LFS+NBS	0.8233	**0.8221**	(BC-2)+FS+SL	
PC5	0.9310	0.9320	LOG+BE+NB	0.9830	0.9840	LOG+FS+NB	0.9867	**0.9863**	LOG+BE+MP	
JM1	0.7327	0.7330	(BC-0.5)+LFS+BAG	0.7342	0.7332	(BC-1)+LFS+BAG	0.7402	**0.7404**	(BC-1)+RS+BAG	
AR1	0.4838	0.5045	(BC-2)+FS+LOG	0.7353	0.7318	(BC-1)+BE+MP	0.5064	**0.7343**	(BC-1)+FS+NBS	
AR3	0.4680	0.5000	LOG+RS+LOG	0.7589	0.7524	(BC-2)+GS+MP	0.5033	**0.7528**	(BC-1)+RS+BN	
AR4	0.7404	0.7094	(BC0.5)+LFS+LOG	0.7969	0.8073	NONE+LFS+NBS	0.8080	**0.8081**	(BC0.5)+LFS+LOG	
AR6	0.4053	0.3612	(BC-0.5)+FS+RF	0.7124	0.7260	(BC-2)+BE+MP	0.7045	**0.7290**	(BC-0.5)+FS+MP	

Table 7: Mutation variable with Crossover set 0.60

	Genetic								
	0.01			0.033			0.1		
DS	Eval	Pred	LS	Eval	Pred	LS	Eval	Pred	LS
CM1	0.7911	**0.8050**	(BC0.5)+LFS+LOG	0.7893	0.7921	(BC0.5)+BF+LOG	0.7611	0.7050	(BC0.5)+RS+LOG
KC3	0.7859	**0.8139**	(BC0.5)+LFS+NBS	0.7195	0.8089	(BC0-0.5)+LFS+LOG	0.6959	0.6139	(BC0.5)+RS+LOG
KC4	0.9066	**0.9051**	(BC0.5)+LFS+LOG	0.8887	0.8790	(BC0-1)+LFS+NB	0.8504	0.8432	(BC0.5)+LFS+MP
MW1	0.8786	**0.8886**	(BC1)+LFS+NB	0.7010	0.7256	(BC1)+FS+NB	0.7130	0.6686	LOG+RS+SL
PC1	0.8384	0.8220	NONE+BE+BAG	0.8255	0.8426	NONE+LFS+BAG	0.8284	**0.8556**	NONE+LFS+BAG
PC2	0.8745	**0.9084**	(BC0-2)+FS+LOG	0.6681	0.7171	(BC0-2)+FS+LOG	0.6645	0.6984	(BC0-1)+LFS+NBS
PC3	0.8223	**0.8236**	(BC0.5)+LFS+MP	0.8246	0.8104	(BC0.5)+FS+MP	0.8211	0.8223	(BC0.5)+LFS+SL
PC4	0.9264	**0.9236**	(BC0-1)+FS+LGB	0.9184	0.9097	(BC0-0.5)+BE+LGB	0.9164	**0.9236**	(BC0.5)+LFS+LOG
KC1	0.8060	0.7082	(BC1)+LFS+BAG	0.8045	**0.8078**	(BC1)+FS+BAG	0.8060	0.7982	(BC1)+RS+BAG
MC1	0.9820	0.9808	NONE+LFS+MP	0.9865	**0.9889**	NONE+LFS+LOG	0.9299	0.9154	NONE+GS+MP
MC2	0.8145	**0.8155**	(BC0.5)+LFS+NBS	0.6996	0.6371	NONE+BE+SL	0.8145	**0.8155**	(BC0-2)+FS+NB
PC5	0.9830	**0.9840**	NONE+LFS+SL	0.9778	0.9743	LOG+LFS+SL	0.9545	0.9576	NONE+GS+SL
JM1	0.7342	**0.7332**	(BC0-1)+LFS+BAG	0.7111	0.7298	(BC0-1)+LFS+MP	0.7065	0.7066	(BC0-1)+LFS+SL
AR1	0.7353	**0.7318**	(BC0-1)+BE+MP	0.4446	0.6029	(BC0-1)+LFS+MP	0.5518	0.4553	(BC0-1)+FS+MP
AR3	0.7589	**0.7524**	(BC0-2)+GS+MP	0.4490	0.4490	(BC0-2)+GS+MP	0.7889	0.7424	NONE+GS+NBS
AR4	0.7969	0.8073	NONE+LFS+NBS	0.7244	**0.8958**	NONE+BE+SL	0.7473	0.7446	(BC0.5)+FS+NB
AR6	0.7124	**0.7260**	(BC0-2)+BE+MP	0.5050	0.4062	(BC0-2)+RS+RF	0.7824	0.7160	LOG+BE+LOG

Table 8: Crossover variable with Mutation set 0.01

	Genetic								
	0.6			0.7			0.9		
DS	Eval	Pred	LS	Eval	Pred	LS	Eval	Pred	LS
CM1	0.7911	0.8050	(BC0.5)+LFS+LOG	0.7811	0.8140	(BC0.5)+LFS+MP	0.8067	**0.8143**	(BC0.5)+LFS+SL
KC3	0.7859	0.8139	(BC0.5)+LFS+NBS	0.7523	**0.8166**	(BC0.5)+RS+NBS	0.7944	0.8023	NONE+LFS+NBS
KC4	0.9066	0.9051	(BC0.5)+LFS+LOG	0.9167	0.9143	(BC1)+RS+LOG	0.9164	**0.9190**	LOG+RS+MP
MW1	0.8786	**0.8886**	(BC1)+LFS+NB	0.6877	0.6454	NONE+BE+NB	0.6824	0.7637	(BC1)+BE+LOG
PC1	0.8384	0.8220	NONE+BE+BAG	0.8307	0.8119	NONE+LFS+BAG	0.8321	**0.8232**	LOG+BE+BAG
PC2	0.8745	**0.9084**	(BC-2)+FS+LOG	0.6808	0.7670	(BC-2)+BE+LOG	0.6911	0.8045	LOG+BF+LOG
PC3	0.8223	**0.8236**	(BC0.5)+LFS+MP	0.6745	0.7021	(BC0.5)+FS+MP	0.8045	0.7833	(BC0.5)+BF+BAG
PC4	0.9264	**0.9236**	(BC-1)+FS+LGB	0.9019	0.9037	(BC1)+LFS+BAG	0.9175	0.9176	LOG+BE+LGB
KC1	0.8060	0.7082	(BC1)+LFS+BAG	0.8034	0.8112	(BC1)+BF+LOG	0.8155	**0.8121**	(BC1)+LFS+BN
MC1	0.9830	**0.9840**	NONE+LFS+MP	0.9611	0.9834	NONE+LFS+BAG	0.9867	0.9832	NONE+RS+NBS
MC2	0.8145	0.8155	(BC0.5)+LFS+NBS	0.6857	0.7340	(BC-2)+RS+MP	0.8021	**0.8262**	(BC-1)+LFS+NBS
PC5	0.9830	0.9840	NONE+LFS+SL	0.9723	0.9746	NONE+RS+MP	0.9825	**0.9887**	NONE+LFS+MP
JM1	0.7342	0.7332	(BC-1)+LFS+BAG	0.7167	0.7232	(BC-1)+LFS+LOG	0.7345	**0.7371**	(BC-1)+LFS+BAG
AR1	0.7353	**0.7318**	(BC-1)+BE+MP	0.4780	0.6227	(BC-2)+BE+MP	0.4835	0.4090	(BC-2)+BE+MP
AR3	0.7589	**0.7524**	(BC-2)+GS+MP	0.4173	0.4425	(BC1)+GS+NBS	0.4764	0.6437	(BC0.5)+BE+MP
AR4	0.7969	0.8073	NONE+LFS+NBS	0.7265	**0.8236**	NONE+LFS+NB	0.7404	0.7416	NONE+LFS+NB
AR6	0.7124	**0.7260**	(BC-2)+BE+MP	0.5206	0.2452	(BC-1)+BE+MP	0.5320	0.6473	(BC-2)+RS+RT

10 CONCLUSIONS AND FUTURE WORK

In all, 864 learning schemes were studied. The most predominant data pre-processing was BoxCox, with a representation of (76.47%) and NONE (23.53%). The most predominant attribute selectors were LFS, with a representation of (64.70%), BE (17.64%), FS (11.64%) and GS (5.88%). Finally, the main learning algorithms were MP (29.41%), BAG, NBS and LOG, with 17.64% each. Other important learning algorithms were NB, SL and LGB, representing 5.88% each. The main pattern found it was BoxCox + LFS + MP.

According to statistical analysis, we did not find a statistically significant difference between our framework and Song's, with a search space of 12 combinations, considering performance. On the other hand, we did find a statistically significant difference between our framework and Song's, with a search space of 864 combinations, considering performance. However we did find a statistically significant difference between our framework and an exhaustive framework, considering runtime.

Finally, we conducted a sensibility analysis of the main genetic operators and results, and we did not find a predominant configuration per data set. According to results, the crossover operator reported for 0.6 (47.05%), 0.7 (11.76%), 0.9 (41.17%). This means that 0.6 and 0.9 were the best configurations for the crossover operator. Based on the mutation operator, the results were 0.01 (76.47%), 0.033 and 0.1 (17.64%) each. This means that 0.01 was the best configuration for the mutation operator. An important conclusion is that there is no specific genetic configuration for all data sets. Thus, it will depend on the characteristics of the domain analyzed. This is a reason why it is important to apply a sensitivity analysis, with the objective of selecting the best possible configuration according to the data sets analyzed.

In the future, we plan to work with other kinds of data sets. For example private data sets or projects. Furthermore, we will like to work with more learning schemes providing more AS, DP and LA techniques. Additionally, we plan to conduct more experimentation with different genetic implementations using other types of implementations (operators). Further, increasing the $N - PASS$ parameter, experimenting with other performance metrics: such as precision, recall, balance, among others. Finally, we will execute an analysis of the treatments (864) in order to find interactions between the factors (learning schemes).

11 ACKNOWLEDGMENTS

This research was supported by University of Costa Rica, National University of Costa Rica and Ministry of Science, Technology and Telecommunications (MICITT). This work has been done in the context of the project "Collaborative Ambient Assisted Living Design (ColoSAAL)" (grant TIN2014-57028-R) supported by the Spanish Ministry for Economy and Competitiveness, the research programme MOSI-AGIL-CM (grant S2013/ICE-3019) supported by the Autonomous Region of Madrid and co-funded by EU Structural Funds FSE and FEDER, and the "Programa de Creación y Consolidación de Grupos de Investigación" (UCM-BSCH GR35/10-A).

References

[1] Erik Arisholm, Lionel C Briand, and Eivind B Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17, 2010.

[2] Pratibha Bajpai and Manoj Kumar. Genetic algorithm an approach to solve global optimization problems. *Indian Journal of computer science and engineering*, 1(3):199–206, 2010.

[3] Anju Bala and Aman Kumar Sharma. A comparative study of modified crossover operators. In *2015 Third International Conference on Image Information Processing (ICIIP)*, pages 281–284. IEEE, 2015. DOI: 10.1109/ICIIP.2015.7414781.

[4] Victor R Basili. Software modeling and measurement: the goal/question/metric paradigm. 1992.

[5] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997. DOI: 10.1016/S0031-3203(96)00142-2.

[6] Cagatay Catal. Performance evaluation metrics for software fault prediction studies. *Acta Polytechnica Hungarica*, 9(4):193–206, 2012.

[7] Cagatay Catal and Banu Diri. Unlabelled extra data do not always mean extra performance for semi-supervised fault prediction. *Expert Systems*, 26(5):458–471, 2009. DOI: 10.1111/j.1468-0394.2009.00509.x.

[8] Cagatay Catal, Ugur Sevim, and Banu Diri. Clustering and metrics thresholds based software fault prediction of unlabeled program modules. In *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*, pages 199–204. IEEE, 2009. DOI: 10.1109/ITNG.2009.12.

[9] Nitesh V Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 853–867. Springer, 2005.

[10] B Terry Compton and Carol Withrow. Prediction and control of ada software defects. *Journal of Systems and Software*, 12(3):199–207, 1990.

[11] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006. DOI: 10.1145/1143844.1143874.

[12] Andre B De Carvalho, Aurora Pozo, and Silvia Regina Vergilio. A symbolic fault-prediction model based on multiobjective particle swarm optimization. *Journal of Systems and Software*, 83(5):868–882, 2010. DOI: 10.1016/j.jss.2009.12.023.

[13] Nader B. Ebrahimi. On the statistical analysis of the number of errors remaining in a software design document after inspection. *Software Engineering, IEEE Transactions on*, 23(8):529–532, 1997.

[14] César Ferri, Peter Flach, and José Hernández-Orallo. Learning decision trees using the area under the roc curve. In *ICML*, volume 2, pages 139–146, 2002.

[15] N Gayatri, S Nickolas, AV Reddy, and R Chitra. Performance analysis of datamining algorithms for software quality prediction. In *Advances in Recent Technologies in Communication and Computing, 2009. ARTCom'09. International Conference on*, pages 393–395. IEEE, 2009. DOI: 10.1109/ARTCom.2009.12.

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

[17] Maurice Howard Halstead. *Elements of software science*, volume 7. Elsevier New York, 1977.

[18] Yue Jiang, Bojan Cuki, Tim Menzies, and Nick Bartlow. Comparing design and code metrics for software quality prediction. In *Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 11–18. ACM, 2008.

[19] Yue Jiang, Bojan Cukic, and Tim Menzies. Can data transformation help in the detection of fault-prone modules? In *Proceedings of the 2008 workshop on Defects in large software systems*, pages 16–20. ACM, 2008. DOI: 10.1145/1390817.1390822.

[20] Yue Jiang, Jie Lin, Bojan Cukic, and Tim Menzies. Variance analysis in software fault prediction models. In *Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on*, pages 99–108. IEEE, 2009. DOI: 10.1109/ISSRE.2009.13.

[21] Taghi M Khoshgoftaar, Kehan Gao, and Naeem Seliya. Attribute selection and imbalanced data: Problems in software defect prediction. In *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, volume 1, pages 137–144. IEEE, 2010.

[22] Huihua Lu and Bojan Cukic. An adaptive approach with active learning in software fault prediction. In *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, PROMISE '12, pages 79–88, New York, NY, USA, 2012. DOI: 10.1145/2365324.2365335. ACM.

[23] Ruchika Malhotra. Comparative analysis of statistical and machine learning methods for predicting faulty modules. *Applied Soft Computing*, 21:286–297, 2014. DOI: 10.1016/j.asoc.2014.03.032.

[24] Ruchika Malhotra. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, 27:504–518, 2015. DOI: 10.1016/j.asoc.2014.11.023.

[25] Ruchika Malhotra and Ankita Jain. Fault prediction using statistical and machine learning methods for improving software quality. *JIPS*, 8(2):241–262. DOI: 10.3745/JIPS.2012.8.2.241, 2012.

[26] Ruchika Malhotra, Arvinder Kaur, and Yogesh Singh. Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines. *International Journal of System Assurance Engineering and Management*, 1(3):269–281, 2010. DOI: 10.1007/s13198-011-0048-7.

[27] Thomas J McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976. DOI: 10.1109/TSE.1976.233837.

[28] 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, Jan 2007. DOI: 10.1109/TSE.2007.10.

[29] Bharavi Mishra and KK Shukla. Impact of attribute selection on defect proneness prediction in oo software. In *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on*, pages 367–372. IEEE, 2011.

[30] JC Munson and TM Khoshgoftaar. Regression modelling of software quality: empirical investigation. *Information and Software Technology*, 32(2):106–114, 1990.

[31] Juan Murillo-Morera, Carlos Castro-Herrera, Javier Arroyo, and Rubén Fuentes-Fernández. An empirical validation of learning schemes using an automated genetic defect prediction framework. pages 222–234, 2016. DOI: 10.1007/978-3-319-47955-2_19.

[32] Juan Murillo-Morera and Marcelo Jenkins. A software defect-proneness prediction framework: A new approach using genetic algorithms to generate learning schemes. In *The 27th International Conference on Software Engineering and Knowledge Engineering, SEKE 2015, Wyndham Pittsburgh University Center, Pittsburgh, PA, USA, July 6-8, 2015*, pages 445–450, 2015. DOI: 10.18293/SEKE2015-99.

[33] Juan Murillo-Morera, Christian Quesada-López, and Marcelo Jenkins. Software fault prediction: A systematic mapping study. *CIBSE 2015 - XVIII Ibero-American Conference on Software Engineering*, pages 446–459, 2015.

[34] Foyzur Rahman and Premkumar Devanbu. How, and why, process metrics are better. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 432–441. IEEE Press, 2013. DOI: 10.1016/j.jss.2009.06.055.

[35] George P Rédei. *Encyclopedia of genetics, genomics, proteomics, and informatics*. Springer Science & Business Media, 2008.

[36] N Rotstan and K Meffert. Jgap: Java genetic algorithms package. *Programming Package*, 2005. DOI: 10.1016/j.jss.2009.06.055.

[37] N. Seliya, T.M. Khoshgoftaar, and J. Van Hulse. Predicting faults in high assurance software. In *High-Assurance Systems Engineering (HASE), 2010 IEEE 12th International Symposium on*, pages 26–34, Nov 2010. DOI: 10.1109/HASE.2010.29.

[38] Naeem Seliya and Taghi M Khoshgoftaar. The use of decision trees for cost-sensitive classification: an empirical study in software quality prediction. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(5):448–459, 2011. DOI: 10.1002/widm.38.

[39] J Shirabad and T Menzies. The promise repository of software engineering databases (2005). *URL http://promise. site. uottawa. ca/SERepository*.

[40] J Sayyad Shirabad and TJ Menzies. The promise repository of software engineering databases. school of information technology and engineering, university of ottawa, canada, 2005. *promise. site. uottawa. ca/SERepository*, 2005.

[41] Pradeep Singh and Shrish Verma. Empirical investigation of fault prediction capability of object oriented metrics of open source software. In *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, pages 323–327. IEEE, 2012.

[42] Yogesh Singh, Arvinder Kaur, and Ruchika Malhotra. Empirical validation of object-oriented metrics for predicting fault proneness models. *Software quality journal*, 18(1):3–35, 2010. DOI: 10.1007/s11219-009-9079-6.

[43] Qinbao Song, Zihan Jia, Martin Shepperd, Shi Ying, and Jin Liu. A general software defect-proneness prediction framework. *Software Engineering, IEEE Transactions on*, 37(3):356–370, 2011. DOI: 10.1109/TSE.2010.90.

[44] Qinbao Song, M. Shepperd, M. Cartwright, and C. Mair. Software defect association mining and defect correction effort prediction. *Software Engineering, IEEE Transactions on*, 32(2):69–82, Feb 2006. DOI: 10.1109/TSE.2006.1599417.

[45] C Srinivas, B Ramgopal Reddy, K Ramji, and R Naveen. Sensitivity analysis to determine the parameters of genetic algorithm for machine layout. *Procedia Materials Science*, 6:866–876, 2014. DOI: 10.1016/j.mspro.2014.07.104.

[46] Huanjing Wang, Taghi M Khoshgoftaar, and Amri Napolitano. Software measurement data reduction using ensemble techniques. *Neurocomputing*, 92:124–132, 2012. DOI: 10.1016/j.neucom.2011.08.040.

[47] Gabriel Winter, J Periaux, M Galan, and P Cuesta. *Genetic algorithms in engineering and computer science*. John Wiley & Sons, Inc., 1996.

[48] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.