A Software Defect-Proneness Prediction Framework: A new approach using genetic algorithms to generate learning schemes

Juan Murillo-Morera
Department of Informatics
National University of Costa Rica
Heredia, Costa Rica
juan.murillo.morera@una.cr

Marcelo Jenkins
Department of Computer Science
University of Costa Rica
San José, Costa Rica
marcelo.jenkins@ecci.ucr.ac.cr

Abstract

Recently, defect prediction software is an important research topic in the software engineering field. The demand for development of good quality software has seen a rapid growth in the last few years. The software measurement data collected during the software development process include valuable information about software projects status, progress, quality, performance, and evolution. The software fault prediction in the early phases of software development can help and guide software practitioners to focus the available testing resources on the weaker areas during the software development. OBJECTIVE: This paper presents an approach that combines three phases: data preprocessing, attribute selector and learning algorithms using a genetic approach and select the best combination. METHOD: The framework is comprised of 1) scheme learning generator. This component evaluates performance of the learning schemes and suggests the best option according to each data set analyzed, 2) defect predictor component builds models according to the evaluated learning schemes and predicts software defects with new data agreed to the constructed model. CONCLUSIONS: The framework has considered more combinations of learning schemes than other proposals which select the model configuration manually, which means that there are more possibilities to find better learning schemes for each data set. The computational processing of the genetic approach was less costly than Song approach. Finally, The Genetic approach presented an improvement of 0.032 equivalent to 3.2% *more than Song approach.*

Index terms— software metrics, learning schemes, genetic algorithms, fault prediction models, software quality.

doi:10.18293/SEKE2015-099

I. INTRODUCTION

Software fault prediction has been an important research topic in the software engineering field for more than 30 years [1]. The software measurement data collected during the software development process include valuable information about software projects status, progress, quality, performance, and evolution. Software fault prediction models is a significant part of software quality assurance and commonly used to detect faulty software modules, based on software measurement data (software metrics) [2], [3], [4].

Current defect prediction that works on: estimating the number of defects remaining in software systems, discovering defect associations, and classifying the defect-proneness of software components, typically into two classes, defect-prone and non defect-prone [1].

The first approach, employs statistical methods to estimate a number of defects or defect density [5], [6]. The prediction result can be used as an important measure for the software developer and can be used to control the software process, for example, decide whether to schedule further inspections or pass the software artifacts to the next development step. The second approach, borrows association rule mining algorithms from the data mining community to reveal software defect associations [7]. The third approach, works classifying software components as defect-prone and non-defect-prone, means of metric based classification: [8] and [2]. Being able to predict which components are more likely to be defect-prone supports better targeted resting resources and therefore improved efficiency. Unfortunately, classification remains a largely unsolved problem. In order to address this, researchers have been using increasingly learning schemes that include data preprocessing, attribute selector and learning algorithms. The main problem is how to select the best learning scheme, according to specific data set?. Actually does not exist a proposal that

uses genetic algorithm with the objective to select the best learning scheme configuration using a specific data set. The learning schemes have an important problem, it is how to select a correct combination of data preprocessing, attribute selection and learning algorithm for a particular data set. This novel framework has the capacity to combine different learning schemes with the objective to find the best solution according to the parameters selected and the evaluation of the performance metrics proposed.

The general objetive of this research is to propose a Defect-Proneness Prediction Framework with two specific components: learning scheme generator and defect predictor.

The remainder of the article is structured as follows. Section 2 presents the background. Section 3 presents the related work. The proposed framework is presented in Section 4. Section 5 genetic approach. Section 6 experimental setup. Finally, Section 7 conclusions and future work.

II. BACKGROUND

A. Metrics

Defect predictors from static code attribute were used and defined by McCabe [9] and Halstead [10]. McCabe and Halstead are **module-based metrics**, where a module is the smallest unit of functionality (In other computational languages, modules may be called **function** or **method**). The static code attributes are useful, easy to use, and widely used.

Useful. The static code attributes have been used for the prediction of software projects with similar characteristics [1], [11]. **Easy to use**. Static code attribute like lines of code and the McCabe/Halstead attribute can be automatically and cheaply collected, even for very large systems. By contrast, other methods, such as manual code reviews, are labor-intensive. Depending on the review methods. **Widely used**. Many researchers use static attribute to guide software quality predictions: [1], [12], [13].

Halstead attribute were derived by Maurice Halstead in 1977. He argued that modules that are hard to read are more likely to be fault prone. Halstead estimates reading complexity by counting the number of operators and operands in a module. A complete reference [11].

An alternative to Halstead attributes, are the complexity attributes proposed by Thomas McCabe in 1976. Unlike, Halstead and McCabe argued that the complexity of pathways, between module symbols is more than just a count of the symbols. A complete reference [11].

B. Data sets

The most frequent data sets used by software fault prediction researchers are: CM1, JM1, KC1, KC2, KC3, KC4, MW1, MC1, MC2, PC1, PC2, PC3, PC4, PC5, AR1, AR3, AR4 and AR6. These data sets have been evaluated respect to metrics, number of attribute, number of modules among other parameters [1].

C. Learning Schemes

The Learning schemes are composed by: data preprocessing, attribute selector and learning algorithms [1]. **Data preprocessing:** It is very important to build learners. The data are pre-processed, such as removing outliers, handling missing values, and discretizing or transforming numeric attribute. **Attribute selection:** It is important when the data set may not have originally been intended for defect prediction. Not all the attributes may be helpful for defect prediction. Attribute selection methods can be categorized as filters or wrappers [15]. **Learning algorithms:** Once attribute selection has been completed, the best attribute subset are processed. Then the data set represents those attribute subset and the learning algorithm are used to build the learner.

III. RELATED WORK

Traditionally, many researchers have explored issues like the relative metrics of McCabe's cyclomatic complexity, Halstead's software science measures, and lines of code counts for building defect predictors. However, Menzies et al. [11], published a study in 2007 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, which, at the time of their research, contained 10 separate data sets. They claimed that "such debates are irrelevant since how the attributes are used to build predictors is much more important than which particular attributes are used" and "the choice of learning method is far more important than which subset of the available data is used for learning"

Song et al. [1] published a study in which they proposed a fault prediction framework based on Menzies study but analyzing only 12 learning schemes. They argued that although "how is more important than which". The choice of which attribute subset is used for learning is not only circumscribed by the attribute subset itself and available data, but also by attribute selectors, learning algorithms, and data preprocessors. It is well known that there is an intrinsic relationship between a learning method and an attribute selection method.

Malhotra [14] published a systematic review in which she proposed as future work "There are very few studies that examine the effectiveness of evolutionary algorithms". She points out "The future studies may focus on the predictive accuracy of evolutionary algorithms for software fault prediction".

The aim of this paper is to build a framework that generates learning schemes using genetic algorithms. Previous works have analyzed relationships between data preprocessing, attribute selection and learning algorithms. They have used a few combinations, mainly because the evaluation of the learning schemes is processed manually, selecting the combinations. The novel proposed framework tries to select the best combination per data set, according to AUC value(maximum value).

IV. PROPOSED FRAMEWORK

It is very important before building defect prediction model(s) to decide which learning schemes should be used to construct the model. Thus, the predictive performance of learning scheme should be determined for future data. This novel framework is based on Song framework [1]. The proposed framework uses the methodology of Song except how this select the learning schemes. The novel framework consists of two components: 1) Learning Schemes Generator and 2) Defect Prediction. Figure 1, contains the details.

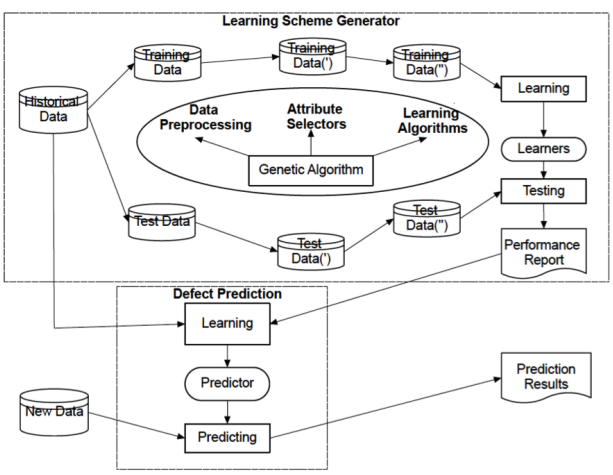

Figure 1. Fault prediction framework with genetic implementation

A. Learning Schemes Generator

The Learning Schemes Generator is a fundamental part of the software defect prediction framework. At this stage, different learning schemes are evaluated, the best one is selected. A Genetic algorithm is used to select the best learning scheme for each data set analyzed based on their AUC. The historical data (represented by 90% of the original data) was divided into training and test data applying a MxN cross-validation based on [1].

The main steps of the Learning Scheme Generator are:

- Each data set is divided into two parts: One part is used as historical data and the other part is viewed as the new data. The historical data are represented by 90% of the original data, while the new data are represented by 10% of the original data.
- The historical data is divided into training set and test set, using a MxN cross-validation.
- The learning scheme elements (data preprocessing, attribute selector and learning algorithm) are selected by a genetic approach considering the fitness function.
- Data preprocessing is applied to both: training and test set. The test set is selected by the genetic algorithm. The result is training data(') and test data(') (see Figure 1).
- Attribute selector is applied to only training set and the best subset of attribute is applied to training and test set. The result is training data(") and test data(") see Figure 1. The attribute subset is computed interactively using a Filter strategy with NxM cross-validation different than Song, who used Wrapper evaluation. The difference is that Wrapper is computationally more costly. This a task of the genetic approach.
- Learning algorithm are build with a training set, and evaluated with a test set. This a task of the genetic approach.

B. Defect Prediction

The defect prediction is part of the proposed framework, consists of predictor construction and defect prediction. The inputs in this stage are: newData, is a data set that represents the new datas. It represents the 10% of the whole data. The other input is the HistoricalData that represents a data set with the other 90% data. Finally, the last input is the learned scheme selected by genetic algorithm. The final results are a log file with two labels: actual value and predicted value.

The main steps of the defect prediction are:

- This component uses the learning scheme selected by genetic algorithm in the previous stage.

- A predictor is build with the selected learning scheme. The whole historical data is used (not apply NxM cross-validation). All the historical data is used to build the predictor, it is expected that the constructed predictor has stronger generalization ability.

- After the predictor is build, new data are preprocessed in the same way as historical data, then the constructed predictor can be used to predict software defect with preprocessed new data.

C. Difference between the approach proposed and Song approach

The approach proposed is based on Song methodology [1]. The contributions of this novel proposal are:

- Song framework only works with 12 learning schemes. The proposed framework works with more combinations. Our maximum search space is: Data preprocessing = 7, attribute selector = 40 and learning algorithms = 41 in total (7*40*41) = 11480. Selecting automatically the best learning scheme per data set, while Song framework selects the learning scheme manually working with backward elimination and forward selection.

- Song framework works with Wrapper in the attribute selection. This computationally is very costly. The proposed framework works with Filter using NxM cross-validation.

- Song framework has a scheme evaluation component. The outcome of this component is a performance report. The proposed framework has a generator of schemes and the outcome is the best scheme learning per data set processed.

V. GENETIC SETUP

The genetic approach to be explained in the following sections: Chromosome, Operators and Fitness Function.

A. Chromosome

The chromosome is represented by a binary chain of 0s and 1s. The representation is a triple of $< DP, AS, LA >$ that genetically is modified. The first part of the chromosome represents the data pre-processing. For the data preprocessing(DP) there are 7 possibilities, represented by a binary chain of $2^3 = 3$ bits. Additionally, for the attribute selector(AS), there are a maximum of 40 metrics (Hasteald, McCabe and LOC), representing a binary chain maximum of $2^6 = 6$ bits. A bit with value 1 represent that this metric is present in the data set, while a bit with value 0 that it is not represented. Finally for the learning algorithms (LA) there are 41 different possibilities, representing a binary chain maximum of $2^6 = 6$ bits.

B. Operators

The operators of selection, reproduction, crossover and mutation used were applied using the following configuration: (Population size = 50, Generations = 100, Crossover probability = 0.6, Mutation probability = 0.1 and Elitism 2%

C. Fitness Function

The Fitness was defined: $f(x) = AUC$ value. AUC value (min 0 - max 1) is defined by X-Axis and Y-Axis. It is an Area Under Curve. Y-Axis is represented by True Positive Rate or Sensitivity and X-Axis is represented by False Positive Rate or (1-Specificity).

VI. EXPERIMENTAL SETUP

A. Data sets

The data sets used were taken from the public NASA MDP repository. This study used 10 data sets: CM1, KC3, MW1, PC1, PC2, PC3, PC4, KC1, MC1 and MC2.

B. Performance Measures

This study has used the metric AUC for the comparison between approaches. Table 1 shows a complete description about this metric.

Table 1. Metrics

Name	Description	Representation
TPR	Sensitivity	$TP/(TP+FN)$
FPR	1- Specificity	$1-(TN/(TN+FP))$
AUC	Area Under Curve	Plot Specificity (X-axis) Plot Sensitivity (Y-axis)

However, AUC was selected for the comparison to respect other approaches [1].

C. Learning Schemes

The learning schemes used in this study were:

- *Data Preprocessing (DP)*: Replace Missing Values, Math Expression, RandomSubset, Remove, Standardize, Numeric transform and Numeric to nominal.

- *Attribute selector (AS)*: This selection is a task of the genetic approach. A subset of metrics or predictors variables are selected according to the genetic selection.

- *Learning Algorithm (LA)*: The families are: Bayes (9), Functions (9), Rules (9) and Trees (14). A complete reference [15]

D. Baseline and Comparison

The Song study [1] (Song-Approach) was selected as baseline for this article. The mean of Backward Selection (BS) algorithm and the mean of Forward Elimination (FE) algorithm were calculated and the best result compared with the Genetic approach (G-Approach). Table 2 shows the comparison results with the same decimal representation [1].

Table 2. AUC-Average

Data set	Attributes	Modules	Song-Approach	G-Approach
CM1	38	344	0.634 (1)	0.728 (3)
KC3	40	200	0.689 (10)	0.679 (17)
MW1	38	759	0.635 (7)	0.697 (2)
PC1	38	264	0.711 (15)	0.727 (8)
PC2	37	1585	0.684 (11)	0.635 (14)
PC3	38	1125	0.687 (18)	0.718 (16)
PC4	38	1399	0.789 (13)	0.829 (6)
KC1	22	2096	0.697 (4)	0.778 (20)
MC1	39	9277	0.828 (19)	0.858 (12)
MC2	40	127	0.654 (5)	0.679 (9)

Table 2 shows that G-Approach had a better performance according to AUC metric. The G-Approach presented an average of 0.732 while Song-Approach presented an average of 0.700. The difference between G-approach and Song-approach has been an improvement of 0.032 equivalent to 3.2%. The order of the runs are represented by tiny numbers (see Table 2).

E. Statistical Analysis and Discussion

Figure 2 shows that Genetic approach is better than Song approach per each data set except PC2 and KC3, where the model proposed presented less performance. The MC1 data set has presented the best performance with AUC = 0.85. Additionally, other datasets where the Genetic approach presented better performance were: MW1, PC1, PC3, PC4, KC1, MC1 and MC2.

The data sets with the worst performance in the Genetic approach were: PC2 with a difference of 0.045 respect to Song approach and KC3 with a difference of 0.01 respect to Song. The data set that presented with the Genetic approach the best performance was CM1 with a difference of 0.094. (see Figure 2).

The order of runs was random. Twenty runs was executed using the model proposed in section IV.

The first factor used was the framework. This factor has been represented by two levels (Genetic and Song). Otherwise, the second factor was the data set and has been represented by ten levels: CM1, KC3, MW1, PC1, PC2, PC3, PC4, KC1, MC1 and MC2.

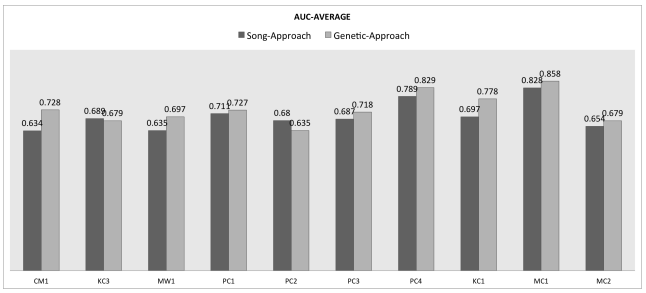

Figure 2. Performance Frameworks

The hypotheses were:

- Hypothesis 1: test the relationship between frameworks according to their performance

  H_{0frm}: Are there significant difference between frameworks respect to AUC? H_{1frm}: Are there not significant difference between frameworks respect to AUC?

- Hypothesis 2: test the relationship between data sets according to their performance H_{0ds}: Are there significant difference between data sets respect to AUC? H_{1ds}: Are there not significant difference between data sets respect to AUC?

Wilcoxon signed rank test was applied for the first hypothesis. A significant difference was found in H_{0frm}. This means that H_{0frm} is rejected and exist a difference statistically significant. The p_{value} reported was $p_{value} = 0.04883 < \alpha = 0.05$. This represented that Genetic approach was better than Song approach, 0.7328 and 0.7004 respectively. Further, the Genetic approach was computationally less costly than Song approach, because the genetic approach has implemented the evaluation with the filter strategy while Song approach has implemented the evaluation with the wrapper strategy.

The second hypothesis was evaluated with an one-way anova study. The first step of this study has been the study of normality, homogeneity of variances and independence assumption. Shapiro-Wilk and Bartlett test were

applied. The results for both test were: normality test, $p_{value} = 0.9051 > \alpha = 0.05$ and homogeneity of variances test $p_{value} = 0.898 > \alpha = 0.05$ (framework) and $p_{value} = 0.90 > \alpha = 0.05$ (data set). The independence principle was assumed. This means not violation of normality assumption. Then the next step was the validation of p_{value} for H_{ds}. A significant difference was found into H_{ds}, this reported a $p_{value} = 0.0496 < \alpha = 0.05$. This means that H_{0ds} is rejected and a Fisher Test can be applied. The Fisher test presented the following results:

Table 3. Group of data sets

Group	DataSets
Group 1	$MC1$
Group 2	$PC4$
Group 3	$PC1, KC1$
Group 4	$CM1, PC3$
Group 5	$KC3, MW1, PC2, MC2$

Table 3 shows the groups with significant difference. All the data sets from different groups have presented significant difference. The group with more data sets is the group-5, and the rest of the groups have been represented with one or two data sets. An important issue is the interval representation. For example, Genetic approach presented an AUC value: min = 0.635, max = 0.858 while Song approach presented an AUC values: min = 0.635, max = 0.828.

VII. CONCLUSIONS AND FUTURE WORK

The framework has included more combinations of learning schemes than other proposals. This means, there are more possibilities to find better learning schemes for each data set. The genetic approach has presented better performance than Song approach in the majority of the cases, representing eight of ten data sets. The data set where the Genetic approach had less performance was PC2 while the data set with more performance was MC1.

The predominant learning schemes were: DP= {Replace Missing Values, Math Expression}, AS={Genetic selection} and LA={Naive Bayes, Decision Tree, Lineal Regression, Boosting, Bagging, Support Vector Machine}.

Another important conclusion has been the size of the data sets. For example MC1 is the data set with more size (9277 modules). This data set represented the best AUC (0.858) in the Genetic approach. A similar situation with KC1, this data set represented the second in size (2096 modules) and the second with the best AUC (0.82). This situation is different in Song approach where the AUC was reported with a value of 0.82 (MC1) and (0.69) KC1 respectively.

As Future work, it is necessary to include more data sets with different size, noise level and imbalance data from public and private repositories. It is very important more experimentation with different parameters configuration and others methods of crossover and mutation that improvement the performance.

VIII. ACKNOWLEDGMENTS

This research was supported by Doctoral Program in Computer Science at the University of Costa Rica. Costa Rican Ministry of Science, Technology and Telecommunications (MICITT).

REFERENCES

[1] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 356–370, 2011.

[2] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "Software measurement data reduction using ensemble techniques," *Neurocomputing*, vol. 92, pp. 124–132, 2012.

[3] 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*, vol. 38, no. 6, pp. 1276–1304, Nov 2012.

[4] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.

[5] J. Munson and T. Khoshgoftaar, "Regression modelling of software quality: empirical investigation," *Information and Software Technology*, vol. 32, no. 2, pp. 106–114, 1990.

[6] N. B. Ebrahimi, "On the statistical analysis of the number of errors remaining in a software design document after inspection," *Software Engineering, IEEE Transactions on*, vol. 23, no. 8, pp. 529–532, 1997.

[7] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "Software defect association mining and defect correction effort prediction," *Software Engineering, IEEE Transactions on*, vol. 32, no. 2, pp. 69–82, Feb 2006.

[8] R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," *Applied Soft Computing*, vol. 21, pp. 286–297, 2014.

[9] T. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, December 1976.

[10] M. Halstead, *Elements of Software Science*. Elsevier, 1977.

[11] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *Software Engineering, IEEE Transactions on*, vol. 33, no. 1, pp. 2–13, Jan 2007.

[12] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 432–441.

[13] R. Malhotra and A. Jain, "Fault prediction using statistical and machine learning methods for improving software quality." *JIPS*, vol. 8, no. 2, pp. 241–262, 2012.

[14] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[15] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.