EasyChair Preprint

№ 3760

Enhancing Deep Learning Capabilities with Genetic Algorithm for Detecting Software Defects

Kajal Tameswar, Geerish Suddul and Kumar Dookhitram

July 6, 2020

Enhancing Deep Learning Capabilities with Genetic Algorithm for Detecting Software Defects

Kajal Tameswar, Geerish Suddul and Kumar Dookhitram

University of Technology, Mauritius (UTM)
Pointe-aux-Sables, Mauritius
kajaltameswar@gmail.com, {g.suddul, kdookhitram}@umail.utm.ac.mu

Abstract. Regardless of existing and well-defined processes, some defects are inevitable, resulting in software performance degradation. The use of traditional machine learning techniques can automate the prediction of software defects. This automated approach significantly improves the quality of the finished product and reduces the cost incurred during development and maintenance stages. The accuracy of artificial neural networks for the automatic prediction of software bugs, can be further enhanced with the use of metaheuristics algorithms. We propose a hybrid approach which combines Genetic Algorithm (GA) and Deep Neural Network (DNN) to better classify software defects. GA is used as a pre-learning phase to automatically optimize the input features for the DNN, as irrelevant variables have a substantial negative impact on the prediction accuracy. Results from experiments using the PROMISE dataset, demonstrates that a DNN consuming optimized features yields better results.

Keywords: machine learning, software bugs, defect prediction, hybrid model, genetic algorithm, deep neural network.

1. 0 Introduction

The technological evolution of software has become an essential and pervasive part of our personal and professional life. Today's wearable technology, implanted medical devices and autonomous driving cars are just a few examples that demonstrate the beginning of a new era of software transformation. The purpose of software defect prediction is to discover major design and programmatic issues which can reduce the huge costs and time imperatives associated with them (*Safial, 2019*). IEEE defined the term fault or bug as 'inappropriate and unexpected behavior in a computer program'. However, due to exponential growth in application software, the assurance of quality in software remains largely an unnoticed subject leading to performance degradation of the industry. As a concern, testing comes into play to find defects or bugs while running a program to produce a zero-defect software (*Chauhan and Singh 2014*). Without

proper testing, a project becomes a definite recipe for disaster that can raise its cost and affects its quality. While bugs persistently continue to worsen the performance of software, the necessity of effective and rapid methods to find software defects is high.

There have been several techniques introduced to reduce the presence of defects in software. For instance, metrics based on object-oriented, traditional and process approaches have broadly been employed in almost every defect prediction model (*Emam et al.2001; Catal and Diri, 2009*). Further applications of software metrics are demonstrated by *Khoshgoftaar et al.2007* with a statistical prediction model based on function-approximation problem analysis and regression. Unfortunately, most of the presented methods fail in providing efficient results, mainly because the architecture of each software is almost unique. As such, prediction models have to take into consideration parameters which are completely different, thereby having difficulty to generalize. To overcome this complicated issue, non-parametric techniques like machine learning and computational intelligence can be considered.

Despite of multiple powerful advances in programming languages and bug detection techniques, software defects affect virtually almost all software products and services. In response to this problem, researchers have widely been studying the topic bug prediction using machine learning approaches which have the potential to leverage the prediction of software bugs (*Puranika et al.2016, Hassan, 2009; Menzies et al.2007; Kim et al.2008; Menzies et al. 2008*). Nevertheless, there still exist many uncertainties with machine learning approaches, as no single techniques have prevailed due to existing imbalanced datasets and lack of formal approaches (*Hassan et al.2018*). We present a novel hybrid approach in this paper using deep neural network along with GA to build an efficient classification-based optimization system for prediction of software defects.

The rest of the paper has been prepared as follows. Section 2.0 provides a short view of related works that have been done in the field of software defects defection. Section 3.0 provides the proposed model. Experimental outcomes and results of the proposed approach are described in Section 4.0. Performance analysis and discussion of results are discussed in section 5.0. Finally, Section 6.0 presents a brief conclusion and future work of the proposed model.

2.0 Literature Review

This section discusses the different software defect prediction techniques identified in the literature. Recently, machine learning approaches have become very popular techniques for defect prediction in software (*Hall et al.2012; Catal and Diri, 2009*). In this context, many algorithms have been designed each having its own data requirements and levels of complexity. Examples comprise of regression algorithms, classification techniques, clustering methods, deep learning and hybrid techniques which is a blend of optimization algorithm and machine learning.

Several supervised classification algorithms such as neural networks, naïve bayes, Support Vector Machines (SVM), linear regression, and K-nearest neighbor, as described by *Perreault et al. 2017* have been used for the prediction and detection of software bugs. On the other hand, regression approaches have been tackled using SVM by *Elish et al. 2008*. SVM has also been used for classification (*Gray et al. 2009*) of defects, which has a special focus on the pre-processing of the input data. *Shivaji et al. 2013* investigated a naïve bayes classification algorithm combined with feature selection module for efficient prediction. Each of the approaches have shown different levels of efficiency, making them difficult to implement. A more efficient deep learning neural network model is presented by *Yang et al.* 2015. Along the same approach the work of *Gondra et al. 2008* demonstrates that labelled datasets with software metrics can help better train neural network models. Another model proposed by *Yang et al. 2006* shows the combination of neural network with radial basis function and Bayesian method.

In unsupervised clustering algorithms, the application of ambiguous datasets has been very popular. For instance, *Bishnu et al. 2012* came up with a k-means clustering model for software bug prediction. Hybrid approaches based on K-means algorithms have been attempted, such as application of the Neural-Gas and Quad Tree techniques for optimum exploration and cluster labelling of real-world datasets (*Rani and Rajalakshmi, 2012; Meenakshi et al.2012*).

Hybrid approaches have the advantage of combining the best of different techniques and hence further improves the accuracy of prediction models. *Azar et al. 2011* developed a model using ant-colony optimization technique for prediction of software bug. Another study (*Rong et al.2016*) proposed a hybrid Support Vector Machine model

combined with the bat search algorithm. *Manjula and Florence, 2018* presented a machine learning based hybrid model by combining genetic optimization algorithm with decision tree algorithm. *Wahono et al. 2014* build a model using neural network based on bagging technique and genetic algorithm for prediction of software bug in order to improve performance.

In regards to the above work, we noticed that software defect prediction models have a high cost associated with it. While some approaches have high processing time other are intricately complex. Genetic Algorithm (GA) has been extensively used in neural network optimization and is known to be successful in achieving optimal solutions. While substantial work has been done regarding neural network parameter optimization using GA in several applications, there has not been sufficient research performed on investigating them in the field of defect prediction. To overcome this problem, we present a hybrid-based model using GA to optimize deep neural network for software defect prediction.

3.0 Proposed Model

The proposed software defect prediction model comprises of a Deep Neural Network (DNN) and Genetic Algorithm (GA). It therefore follows a two-fold approach, as below:

i) Application of GA for feature optimization.
ii) Application of DNN for classification purpose.

3.1 Genetic Algorithm

Genetic Algorithm is a metaheuristic evolutionary algorithm based on the principle of selection and mutation. In our context, GA is applied for the purpose of searching the parameter space, finding the global optimum solution and optimizing the weight and threshold of the neural network effectively (*Suzuki et al.2013*). The parameters that has been used for the implementation of GA were set as: size of population = 100; number of generations = 50; probability of crossover = 0.5; and mutation probability = 0.2.

3.1.1 Deep Neural Network

This section defines the Deep Neural Network used for this study related to prediction of software defects. Deep neural network is useful for the learning of effective features

and discriminative patterns in nature, especially for software bug prediction (*Yang et al.2015*). DNN can also be applied to unlabeled datasets. In our model, we used one input layer and 10 hidden layers to produce the output.

3.1.2 Hybrid Intelligence of Genetic Algorithm with Deep Neural Network

The fundamental ideologies of GA are to generate an initial population of chromosomes followed by selection and crossover in order to achieve effective population having the fittest chromosome (optimal value) among them. Figure 1 below shows the proposed architecture, involving the steps required to build the hybrid predictive model using GA together with DNN.

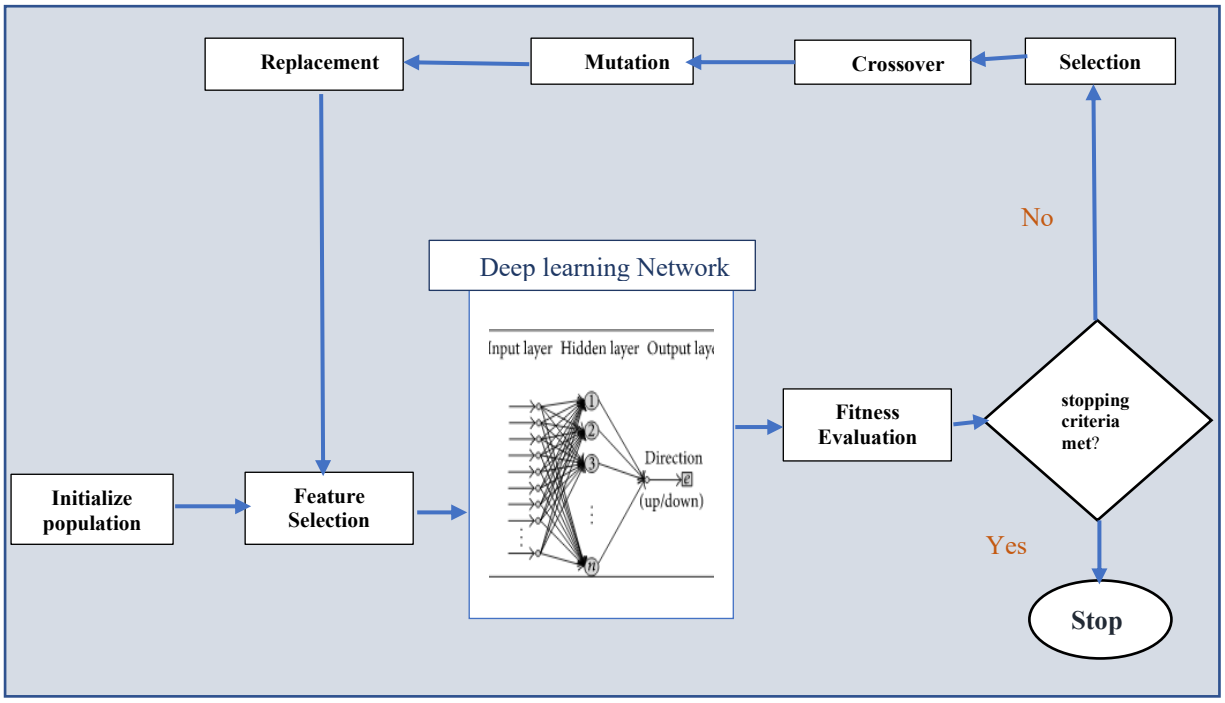

Fig 1: Steps involving building process of GA and DNN.

4.0 Experimental Study and Results

The experimental studies carried out for this proposed approach is described in this section. The hybrid DNN+GA model has been developed using Python packages (Tensor Flow, Scikit-Learn and Keras). TensorFlow is used to train and calculate accuracy of the prediction model. Scikit-Learn is used to read and split the dataset for

training and testing purpose. Keras provides the possibility to speed up experimentation cycles on CPU and GPU. All experiments have been conducted using a laptop consisting of the following configuration: Corei7-6500U CPU, x64 based processor and 16 GB RAM. The PROMISE datasets have been used for training and prediction.

4.1 Dataset details

In PROMISE repository consists of five primary datasets, namely: AR1, AR3, AR4, AR5, AR6. Since they are all related having similar attributes (e.g. loc, comment loc, cyclomatic complexity), we have decided to combine the dataset altogether. The dataset consists of 29 features, and 1050 records, out of which 70% will be used for training and the remaining 30% used for testing. A random state of 65 is used to ensure that each experiment splits the dataset with the same record in every set to acquire appropriate calculation of prediction accuracy for the model.

The datasets are categorized as follows:

i) LOC counts (total_loc, blank_loc, comment_loc, code_and_comment_loc, executable_loc, unique_operands, unique_operators, total_operands, total_operators): Defines numbers of lines of code

ii) Halstead (vocabulary, length, volume, level, difficulty, effort, error time): Based on number of operators and operands

iii) McCabe (cyclomatic_complexity, cyclomatic_density, decision_density, design_complexity, design_density, normalized_cyclomatic_complexity, formal_parameters): This keeps a measure of the number of possible alternative paths through the code

iv) Others (branch_count, decision_count, call_pairs, condition_count, multiple_condition_count)

4.1.1 Fitness Evaluation using performance evaluation metrics

For the performance of the defect prediction model, the following metrics have been used using these annotations which are as follows:

TP = True Positive, FP = False Positive, TN = True Negative, and FN =False Negative.

Metric	Description	Formula
Accuracy	Used for the determination of chromosomes selection and for performance measurement of the hybrid prediction model	(TP+TN)/Total number of samples used
Recall	The percentage result that have correctly been classified by our algorithm	TP/ (TP+FN)
Precision	Defined as the proportion of occurrences predicted as defective which actually are defective	TP/(TP+FP)

Table 1: Evaluation metrics.

5.0 Performance analysis and discussion of results

5.1 Experimental scenario 1

In first instant, experiments were conducted whereby only the DNN has been taken into consideration. These experiments are conducted for AR1, AR3, AR4, AR5, AR6 (combined dataset) with all 29 attributes in the dataset. The efficiency of the DNN prediction model is evaluated and presented statistically in table 2 and in a confusion matrix in table 3. The time taken to run the algorithm was 31. 34 seconds

Table 2: Result for statistical performance analysis using Deep Neural Network (DNN).

Precision	Recall	Accuracy
0.895	0.895	87.21%

Table 3: Confusion matrix using Deep Neural Network (DNN).

Actual class	Predicted Class	
	Defective	Non-Defective
Defective	5	17
Non-Defective	7	243

Figure 2 depicts the ROC curve that shows the performance of the classification model by plotting the true positive and false positive rate; achieving an accuracy of 92.21% for DNN.

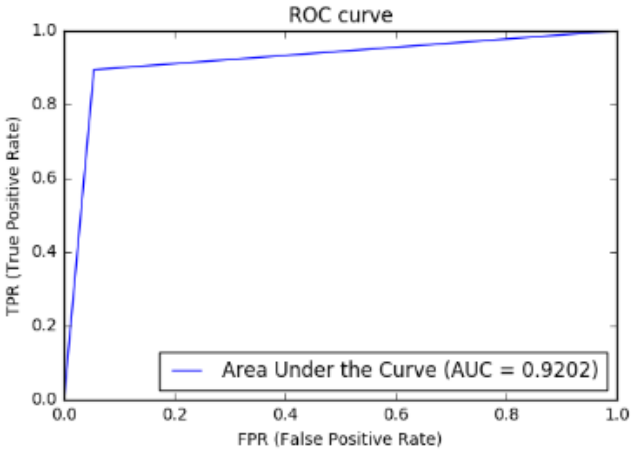

Fig.2 ROC analysis curve.

5.1.1 Experimental scenario 2

Secondly, using the same settings and configurations, experiments has been conducted using the proposed hybrid model, and the results are presented in Table 4 below.

The attributes that had been considered for GA to perform feature selection on DNN was LOC counts and Halstead only. The time taken to run the algorithm was around 12 hours.

Table 4: Result for statistical performance analysis using proposed hybrid approach (DNN+GA).

Precision	Recall	Accuracy
0.896	0.896	92.21%

Table 5: Confusion matrix using Deep Neural Network (DNN) and GA.

Actual class	Predicted Class	
	Defective	Non-Defective
Defective	3	5
Non-Defective	6	258

Figure 3 depicts the ROC analysis curve which takes into consideration both the true positive rate and false positive rate where proposed hybrid approach illustrates better performance when compared to DNN with an accuracy rate of 95%.

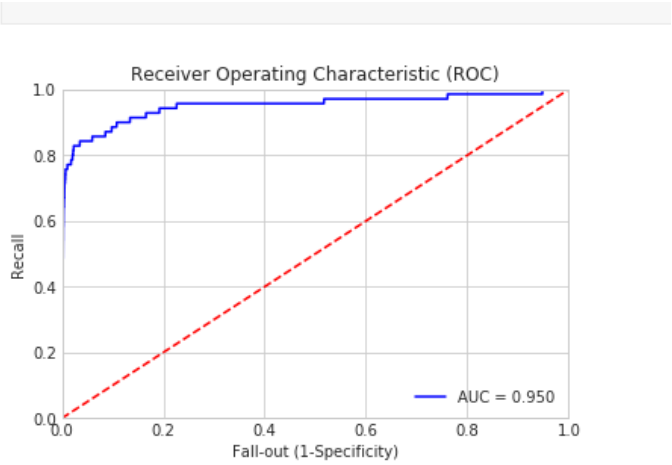

Fig 3: ROC analysis curve.

The result of experimental study shows that the proposed hybrid approach provides reliable performance that can be used for software defect prediction model. The result

produced for the hybrid model has achieved accuracy of 92.21 % while DNN an accuracy of 87.21%.

6.0 Conclusion and Future work

In recent years, early prediction of software defects methods along with their problems and applications are emerging rapidly. This paper presents a hybrid approach for software defect prediction using Deep Neural Network (DNN) classification scheme combined with Genetic Algorithm (GA) using benchmark dataset from PROMISE repository. The performance of this hybrid approach when compared with a conventional DNN shows an increase of around 5% with regards to prediction accuracy.

Future research is highly applicable for this current study where this methodology implemented can be improved by using real-time application datasets. Furthermore, there are some requirements to consider such as overfitting phenomena and noise factors when designing the neural network. Thus, parameters of the learning functions for the neural network should be selected properly for better optimization of hyper parameters in the networks. In addition, control parameters like crossover rate and mutation rate of genetic algorithm should be taken into consideration in order to derive suitable combinations to enhance performance of the model.

References

1. A. E. Hassan. Predicting faults using the complexity of code changes. In Proceedings of the 31st International Conference on Software Engineering, pages 78–88. IEEE Computer Society, 2009.

2. Azar, D., Vybihal, J.: An ant colony optimization algorithm to improve software quality prediction models: case of class stability.Inf. Softw. Technol. **53**(4), 388–393,2011

3. Ayon, Safial.: Neural Network based Software Defect Prediction using Genetic Algorithm and Particle Swarm Optimization. 1-4. 10.1109/ICASERT.2019.8934642,2019

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

5. Bishnu, P.S. and Bhattacherjee, V., Software fault prediction using Quad Tree-based K-means clustering algorithm, IEEE Transaction in Knowledge Data Engineering 24(6), pp.1146–1150, 2012.

6. Catal, C., Diri, B.: A systematic review of software fault prediction studies, *Expert Syst. Appl.*, 2009, 36 (4), pp. 7346–7354

7. Elish, K.O and Elish, M.O.: Predicting defect-prone software modules using support vector machines. J. Syst. Softw. 81(5), pp. 649–660, 2008

8. F. Hassan1, S. Farhan2, M. A. Fahiem3, H. Tauseef,2018, A Review on Machine Learning Techniques for Software Defect Prediction, Technical Journal, University of Engineering and Technology (UET) Taxila, Pakistan Vol. 23(2), pp. 2313-7770, 2018.

9. Gondra, I., Applying machine learning to software fault-proneness prediction. Journal of System Software. 81(2), 186–195, 2008.

10. Gray, D., Bowes, D., Davey, N., Sun, Y., Christianson, B., Using the support vector machine as a classification method for software defect prediction with static code metrics. Engineering Applications of Neural Networks, pp. 223–234. Springer, Berlin, 2009.

11. Hall, T., Beecham, S., Bowes, D., A systematic literature review on fault prediction performance in software engineering', *IEEE Transaction Software Engineering*, 38 (6), pp. 1276–1304, 2012.

12. M. M. R. Henein, D. M. Shawky, and S. K. Abd-El-Hafiz, "Clustering-based Under-sampling for Software Defect Prediction," 13th International Conference on Software Technologies (ICSOFT), pp. 185 – 193, 2018

13. T. M. Khoshgoftaar and K. Gao, "Count Models for Software Quality Estimation," in IEEE Transactions on Reliability, vol. 56, no. 2, pp. 212-222, June 2007.

14. Perreault, L., Berardinelli S., Izurieta C. and Sheppard J., Using Classifiers for Software Defect Detection. 26th International Conference on Software Engineering and Data Engineering (SEDE), 2017

15. Rasneet K. C. and Iqbal S., Latest Research and Development on Software Testing Techniques and Tools, International Journal of Current Engineering and Technology, 4(4), 2014

16. S. C. Yusta, "Different metaheuristic strategies to solve the feature selection problem," *Pattern Recognit. Lett.*, vol. 30, no. 5, pp. 525–534, 2009.

17. S. Kim, E. J. Whitehead, and Y. Zhang. Classifying software changes: Clean or buggy? Software Engineering, IEEE Transactions on, 34(2):181–196, 2008.

18. S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," Proc. of the International Conference on Software Engineering, pp. 297-308, May 14 - 22, 2016.

19. Shivaji, S., James Whitehead, E., Akella, R., Kim, S.: Reducing features to improve code change-based bug prediction. IEEE Transaction on Software Engineering, 39(4), 552–569, 2013.

20. Suzuki, M., Tsuruta, S., Knauf, R.: Structural diversity for genetic algorithms and its use for creating individuals. In: IEEE Congress on Evolutionary Computation, Cancun, pp. 783–788 (2013)

21. Puranika S., Deshpandea P., Chandrasekaran K., A Novel Machine Learning Approach for Bug Prediction, 6th International Conference on Advances In Computing & Communications, ICACC, 6-8 September 2016.

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

23. Menzies T., Turhan B., Bener A., Gay G., Cukic B., and JiangY., Implications of ceiling effects in defect predictors. In Proceedings of the 4th international workshop on Predictor models in software engineering, pp. 47–54. ACM, 2008.

24. Wahono, R.S., Herman, N.S., Ahmad, S.: Neural network parameter optimization based on genetic algorithm for software defect prediction. Adv. Sci. Lett. **20**, 1951–1955 (2014)

25. X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in QRS'15: Proc. of the International Conference on Software Quality, Reliability and Security, 2015.

26. Yang, X., Lo, D., Xia, X., Zhang, Y., Sun, J., Deep learning for just-in-time defect prediction. IEEE International Conference on Software Quality, Reliability and Security (QRS15), pp. 17-26, 2015.