Copyright © 2014 American Scientific Publishers
All rights reserved
Printed in the United States of America

Advanced Science Letters
Vol. 20, 1951–1955, 2014

Neural Network Parameter Optimization Based on Genetic Algorithm for Software Defect Prediction

Romi Satria Wahono[1,2], Nanna Suryana Herman[2], Sabrina Ahmad[2]

[1]Faculty of Computer Science, Dian Nuswantoro University, Indonesia
[2]Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka

Software fault prediction approaches are much more efficient and effective to detect software faults compared to software reviews. Machine learning classification algorithms have been applied for software defect prediction. Neural network has strong fault tolerance and strong ability of nonlinear dynamic processing of software defect data. However, practicability of neural network is affected due to the difficulty of selecting appropriate parameters of network architecture. Software fault prediction datasets are often highly imbalanced class distribution. Class imbalance will reduce classifier performance. A combination of genetic algorithm and bagging technique is proposed for improving the performance of the software defect prediction. Genetic algorithm is applied to deal with the parameter optimization of neural network. Bagging technique is employed to deal with the class imbalance problem. The proposed method is evaluated using the datasets from NASA metric data repository. Results have indicated that the proposed method makes an improvement in neural network prediction performance.

Keywords: Software Defect Prediction, Neural Network, Genetic Algorithm, Bagging Technique

1. INTRODUCTION

Software defects or software faults are expensive in quality and cost. The cost of capturing and correcting defects is one of the most expensive software development activities [1]. Unfortunately, industrial methods of manual software reviews and testing activities can find only 60% of defects [2].

Recent studies show that the probability of detection of fault prediction models may be higher than the probability of detection of software reviews. Menzies et al. found defect predictors with a probability of detection of 71 percent [3]. This is markedly higher than other currently used industrial methods such as manual code reviews. Therefore, software defect prediction has been an important research topic in the software engineering field, especially to solve the inefficiency and ineffectiveness of existing industrial approach of software testing and reviews.

Classification algorithm is a popular machine learning approach for software defect prediction. It categorizes the software code attributes into defective or not defective, which is collected from previous development projects. Classification algorithm able to predict which components are more likely to be defect-prone supports better targeted testing resources and therefore, improved efficiency. If an error is reported during system tests or from field tests, that module's fault data is marked as 1, otherwise 0. For prediction modeling, software metrics are used as

[1] Email: romi@brainmatics.com

 doi:10.1166/asl.2014.5641

independent variables and fault data is used as the dependent variable [4]. Various types of classification algorithms have been applied for predicting software defect, including logistic regression [5], decision trees [6], neural network [8], and naive bayes [9].

Neural network (NN) has strong fault tolerance and strong ability of nonlinear dynamic processing of software fault data, but practicability of neural network is limited due to difficulty of selecting appropriate parameters of network architecture, including number of hidden neuron, learning rate, momentum and training cycles [10]. Rule of thumb or trial-and-error methods are used to determine the parameter settings for NN architectures. However, it is difficult to obtain the optimal parameter settings for NN architectures [11].

On the other hand, software defect datasets have an imbalanced nature with very few defective modules compared to defect-free ones [12]. Imbalance can lead to a model that is not practical in software defect prediction, because most instances will be predicted as non-defect prone [13]. Learning from imbalanced class of dataset is difficult. Class imbalance will reduce or boost classifier performance [14]. The balance of on which models are trained and tested is acknowledged by a few studies as fundamental to the reliability of models [15].

In this research, we propose the combination of genetic algorithm (GA) and bagging technique for improving the accuracy of software defect prediction. GA is applied to deal with the parameter optimization of NN, and bagging technique is employed to deal with the class imbalance problem. GA is chosen due to the ability to find a solution in the full search space and use a global search ability, which significantly increasing the ability of finding high-quality solutions within a reasonable period of time [16]. Bagging technique is chosen due to the effectiveness in handling class imbalance problem in software defect dataset [17] [18].

This paper is organized as follows. In section 2, the related works are explained. In section 3, the proposed method is presented. The experimental results of comparing the proposed method with others are presented in section 4. Finally, our work of this paper is summarized in the last section.

2. RELATED WORKS

The problem of NN is that the number of parameters has to be determined before any training begins. There is no clear rule to optimize them, even though these parameters determine the success of the training process. Thus, it is well known that NN generalization performance depends on a good setting of the parameters. Researchers have been working on optimizing the NN parameters. Wang and Huang [19] has presented an optimization procedure for the GA-based NN model, and applied them to chaotic time series problems. By reevaluating the weight matrices, the optimal topology settings for the NN have been obtained using a GA approach. A particle-swarm-optimization-based approach is proposed by Lin et al. [11] to obtain the suitable parameter settings for NN, and to select the beneficial subset of features which result in a better classification accuracy rate. Then, they applied the proposed method to 23 different datasets from UCI machine learning repository.

However, GA has been extensively used in NN optimization and is known to achieve optimal solutions fair successfully. Previous studies shows that the NN model combined with GA is more effective in finding the parameters of NN than trial-and-error method, and they had been used in a variety of applications [20] [21] [22]. While considerable work has been done for NN parameter optimization using GA in a variety applications, limited research can be found on investigating them in the software defect prediction field.

The class imbalance problem is observed in various domain, including software defect prediction. Several methods have been proposed in literature to deal with class imbalance: data sampling, boosting and bagging. Data sampling is the primary approach for handling class imbalance, and it involves balancing the relative class distributions of the given dataset. There are two types of data sampling approaches: undersampling and oversampling. Boosting is another technique which is very effective when learning from imbalanced data. Seiffert et al. [23] show that boosting performs very well. Bagging techniques generally outperform boosting, and hence in noisy data environments, bagging is the preferred method for handling class imbalance [24]. In the previous works, Wahono et al. have integrated bagging technique and GA based feature selection for software defect prediction. Wahono et al. show that the integration of bagging technique and GA based feature selection is effective to improve classification performance significantly.

In this research, we combine GA for optimizing the NN parameters and bagging technique for solving the class imbalance problem, in the context of software defect prediction. While considerable work has been done for NN parameter optimization and class imbalance problem separately, limited research can be found on investigating them both together, particularly in the software defect prediction field.

3. PROPOSED NN GAPO+B METHOD

We propose a method called NN GAPO+B, which is short for an integration of GA based NN parameter optimization and bagging technique, to achieve better prediction performance of software defect prediction. Figure 1 shows an activity diagram of the proposed NN GAPO+B method.

The aim of GA is to find optimum solution within the potential solution set. Each solution set is called as population. Populations are composed of vectors, namely, chromosome or individual. Each item in the vector is called as gene. In the proposed method, chromosomes represent

NN parameters, including learning rate, momentum and training cycles. The basic process of GA is follows:

1. Randomly generate initial population
2. Estimate the fitness value of each chromosome in the population.
3. Perform the genetic operations, including the crossover, the mutation and the selection
4. Stop the algorithm if termination criterion is satisfied; return to Step 2 otherwise. The termination criterion is the pre-determined maximum number

As shown in Figure 1, input dataset includes training dataset and testing dataset. NN parameters, including, learning rate, momentum and training cycles are selected and optimized, and then NN are trained by training set with selected parameters. Bagging technique [25] was proposed to improve the classification by combining classifications of randomly generated training sets. The bagging classifier separates a training set into several new training sets by random sampling, and builds models based on the new training sets. The final classification result is obtained by the voting of each model.

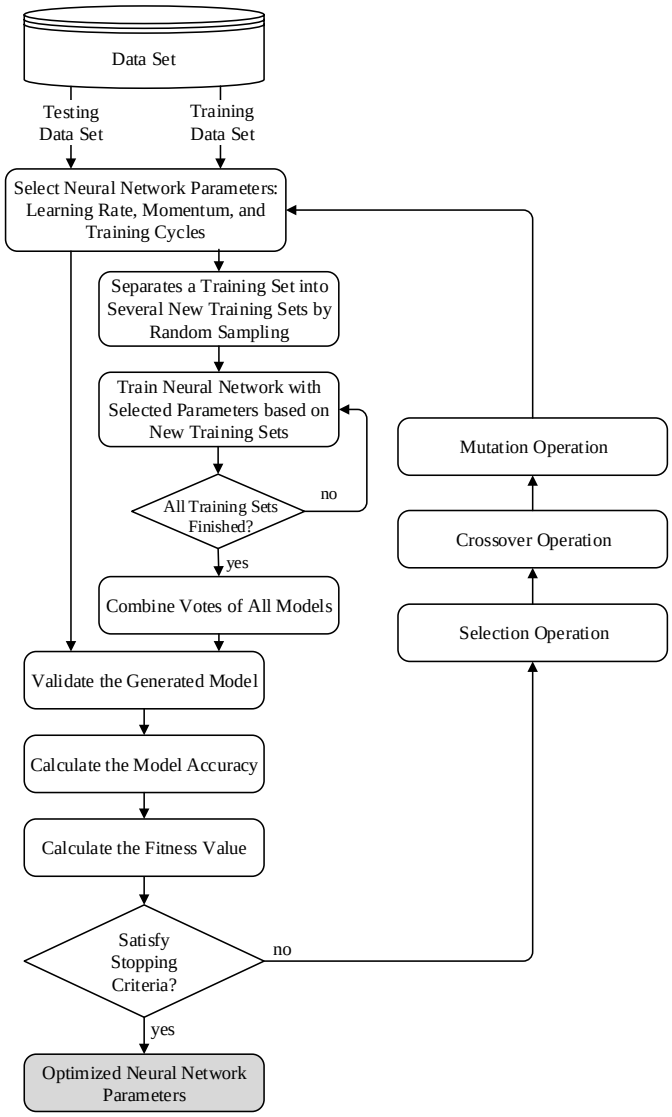

Fig. 1. Activity Diagram of NN GAPO+B Method

Classification accuracy of NN is calculated by testing set with selected parameters. Classification accuracy of NN, the selected parameters and the parameter cost are used to construct a fitness function. Every chromosome is evaluated by the following fitness function equation.

$$fitness = W_A \times A + W_P \times \left(S + \left(\sum_{i=1}^{n} C_i \times P_i \right) \right)^{-1}$$

where A is classification accuracy, W_A is weight of classification accuracy, P_i is parameter value, W_P is parameter weight, C_i is parameter cost, S is setting constant of avoiding that denominator reaches zero.

When ending condition is satisfied, the operation ends and the optimized NN parameters are produced. Otherwise, the process will continue with the next generation operation. The proposed method searches for better solutions by genetic operations, including crossover, mutation and selection.

4. EXPERIMENTAL RESULTS

The experiments are conducted using a computing platform based on Intel Core i7 2.2 GHz CPU, 16 GB RAM, and Microsoft Windows 7 Professional 64-bit with SP1 operating system. The development environment is Netbeans 7 IDE, Java programming language, and RapidMiner 5.2 library.

Table 1. NASA MDP Datasets and the Code Attributes

Code Attributes		NASA MDP Dataset								
		CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
LOC counts	LOC_total	√	√	√	√	√	√	√	√	√
	LOC_blank	√	√	√	√	√	√		√	√
	LOC_code_and_comment	√	√	√	√	√	√	√	√	√
	LOC_comments	√	√	√	√	√	√	√	√	√
	LOC_executable	√	√	√	√	√	√	√	√	√
	number_of_lines	√		√	√	√	√	√	√	√
Halstead	content	√	√	√	√	√	√	√	√	√
	difficulty	√	√	√	√	√	√	√	√	√
	effort	√	√	√	√	√	√	√	√	√
	error_est	√	√	√	√	√	√	√	√	√
	length	√	√	√	√	√	√	√	√	√
	level	√	√	√	√	√	√	√	√	√
	prog_time	√	√	√	√	√	√	√	√	√
	volume	√	√	√	√	√	√	√	√	√
	num_operands	√	√	√	√	√	√	√	√	√
	num_operators	√	√	√	√	√	√	√	√	√
	num_unique_operands	√	√	√	√	√	√	√	√	√
	num_unique_operators	√	√	√	√	√	√	√	√	√
McCabe	cyclomatic_complexity	√	√	√	√	√	√	√	√	√
	cyclomatic_density	√		√	√	√	√	√	√	√
	design_complexity	√	√	√	√	√	√	√	√	√
	essential_complexity	√	√	√	√	√	√	√	√	√
Misc.	branch_count	√	√	√	√	√	√	√	√	√
	call_pairs	√		√	√	√	√	√	√	√
	condition_count	√		√	√	√	√	√	√	√
	decision_count	√		√	√	√	√	√	√	√
	decision_density	√		√	√	√	√	√	√	√
	edge_count	√		√	√	√	√	√	√	√
	essential_density	√		√	√	√	√	√	√	√
	parameter_count	√		√	√	√	√	√	√	√
	maintenance_severity	√		√	√	√	√	√	√	√
	modified_condition_count	√		√	√	√	√	√	√	√
	multiple_condition_count	√		√	√	√	√	√	√	√
	global_data_complexity			√	√					
	global_data_density			√	√					
	normalized_cyclo_complx	√		√	√	√	√	√	√	√
	percent_comments	√		√	√	√	√	√	√	√
	node_count	√		√	√	√	√	√	√	√
Programming Language		*C*	*C++*	*Java*	*C*	*C*	*C*	*C*	*C*	*C*
Number of Code Attributes		*37*	*21*	*39*	*39*	*37*	*37*	*36*	*37*	*37*
Number of Modules		*344*	*2096*	*200*	*127*	*264*	*759*	*1585*	*1125*	*1399*
Number of fp Modules		*42*	*325*	*36*	*44*	*27*	*61*	*16*	*140*	*178*
Percentage of fp Modules		*12.21*	*15.51*	*18*	*34.65*	*10.23*	*8.04*	*1.01*	*12.44*	*12.72*

In this experiments, 9 software defect datasets from NASA MDP [26] are used. Individual attributes per dataset, together with some general statistics and descriptions, are given in Table 1. These datasets have various scales of line of code (LOC), various software modules coded by several different programming languages, including C, C++ and Java, and various types of code metrics, including code size, Halstead's complexity and McCabe's cyclomatic complexity.

The state-of-the-art stratified 10-fold cross-validation for learning and testing data are employed. This means that we divided the training data into 10 equal parts and then performed the learning process 10 times. We employ the stratified 10-fold cross validation, because this method has become the standard method in practical terms. Some tests have also shown that the use of stratification improves results slightly [27]. Area under curve (AUC) is used as an accuracy indicator to evaluate the performance of classifiers in our experiments. Lessmann et al. [10] advocated the use of the AUC to improve cross-study comparability. The AUC has the potential to significantly improve convergence across empirical experiments in software defect prediction, because it separates predictive performance from operating conditions, and represents a general measure of predictiveness.

First of all, we conducted experiments on 9 NASA MDP datasets by using back propagation NN classifier. The experimental results are reported in Table 2 and Figure 2. NN model perform excellent on PC2 dataset, good on PC4 dataset, fairly on CM1, KC1, MC2, PC1, PC3 datasets, but unfortunately poorly on KC3 and MW1 datasets perform.

Table 2. AUC of NN Model on 9 Datasets

Classifiers	CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
NN	0.713	0.791	0.647	0.71	0.625	0.784	0.918	0.79	0.883

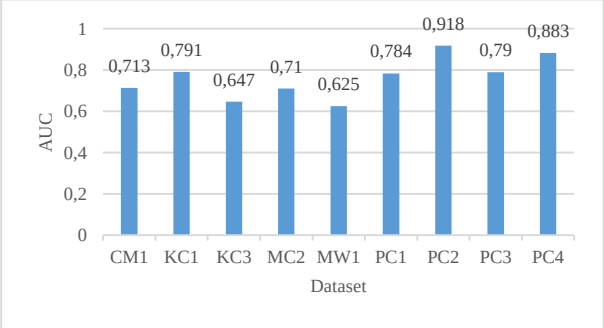

Fig. 2. AUC of NN Model on 9 Datasets

In the next experiment, we implemented NN GAPO+B method on 9 NASA MDP datasets. The experimental result is shown in Table 3 and Figure 3. The improved model is highlighted width boldfaced print. NN GAPO+B model perform excellent on PC2 dataset, good on PC1 and PC4 datasets, and fairly on other datasets. Results show that there were no poor results when the NN GAPO+B model applied.

Table 3. AUC of NN GAPO+B Model on 9 Datasets

Classifiers	CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
NN GAPO+B	**0.744**	**0.794**	**0.703**	**0.779**	**0.76**	**0.801**	**0.92**	**0.798**	0.871

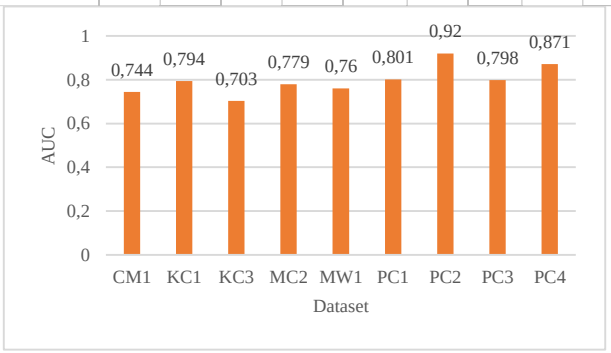

Fig. 3. AUC of NN GAPO+B Model on 9 Datasets

Table 4 and Figure 4 show AUC comparisons of NN model tested on 9 NASA MDP datasets. As shown in Table 4 and Figure 4, although PC4 dataset have no improvement on accuracy, almost all dataset (CM1, KC1, KC3, MC2, MW1, PC1, PC2, PC3) that implemented NN GAPO+B method outperform the original method. It indicates that the integration of GA based NN parameter optimization and bagging technique is effective to improve classification performance of NN significantly.

Table 4. AUC Comparisons of NN Model and NN GAPO+B Model

Classifiers	CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
NN	0.713	0.791	0.647	0.71	0.625	0.784	0.918	0.79	0.883
NN GAPO+B	**0.744**	**0.794**	**0.703**	**0.779**	**0.76**	**0.801**	**0.92**	**0.798**	0.871

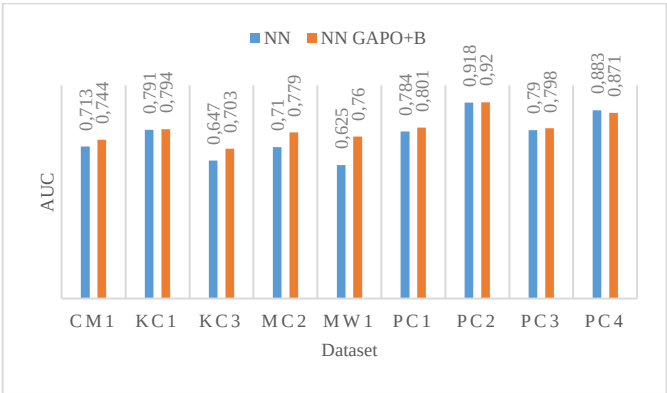

Fig. 4. AUC Comparisons of NN Model and NN GAPO+B Model

Finally, in order to verify whether a significant difference between NN and the proposed NN GAPO+B method, the results of both methods are compared. We performed the statistical t-Test (*Paired Two Sample for Means*) for pair of NN model and NN GAPO+B model on each dataset. In statistical significance testing the *P*-value is the probability of obtaining a test statistic at least as extreme as the one that was actually observed, assuming that the null hypothesis is true. One often "rejects the null hypothesis" when the *P*-value is less than the predetermined significance level (α), indicating that the observed result would be highly unlikely under the null

hypothesis. In this case, we set the statistical significance level (α) to be 0.05. It means that no statistically significant difference if P-value > 0.05.

The result is shown in Table 5. P-value is 0.0279 ($P < 0.05$), it means that there is a statistically significant difference between NN model and NN GAPO+B model. We can conclude that the integration of bagging technique and GA based NN parameter optimization achieved better performance of software defect prediction.

Table 5. Paired Two-tailed t-Test of NN Model and NN GAPO+B Model

	Variable 1	Variable 2
Mean	0.762333333	0.796666667
Variance	0.009773	0.004246
Observations	9	9
Pearson Correlation	0.923351408	
Hypothesized Mean Difference	0	
df	8	
t Stat	-2.235435933	
P	**0.02791077**	

5. CONCLUSION

A combination of genetic algorithm and bagging technique is proposed for improving the performance of the software defect prediction. Genetic algorithm is applied to deal with the parameter optimization of neural network. Bagging technique is employed to deal with the class imbalance problem. The proposed method is applied to 9 NASA MDP datasets with context of software defect prediction. Experimental results show us that the proposed method achieved higher classification accuracy. Therefore, we can conclude that proposed method makes an improvement in neural network prediction performance.

REFERENCES

[1] C. Jones and O. Bonsignour, *The Economics of Software Quality*. Pearson Education, Inc., 2012.

[2] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz, "What we have learned about fighting defects," in *Proceedings Eighth IEEE Symposium on Software Metrics 2002*, 2002, pp. 249–258.

[3] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Autom. Softw. Eng.*, vol. 17, no. 4, pp. 375–407, May 2010.

[4] C. Catal, "Software fault prediction: A literature review and current trends," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 4626–4636, Apr. 2011.

[5] G. Denaro, "Estimating software fault-proneness for tuning testing activities," in *Proceedings of the 22nd International Conference on Software engineering - ICSE '00*, 2000, pp. 704–706.

[6] T. M. Khoshgoftaar, N. Seliya, and K. Gao, "Assessment of a New Three-Group Software Quality Classification Technique: An Empirical Case Study," *Empir. Softw. Eng.*, vol. 10, no. 2, pp. 183–218, Apr. 2005.

[7] B.-J. Park, S.-K. Oh, and W. Pedrycz, "The design of polynomial function-based neural network predictors for detection of software defects," *Inf. Sci. (Ny).*, Jan. 2011.

[8] J. Zheng, "Cost-sensitive boosting neural networks for software defect prediction," *Expert Syst. Appl.*, vol. 37, no. 6, pp. 4537–4543, Jun. 2010.

[9] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[10] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, Jul. 2008.

[11] S.-W. Lin, S.-C. Chen, W.-J. Wu, and C.-H. Chen, "Parameter determination and feature selection for back-propagation network by particle swarm optimization," *Knowl. Inf. Syst.*, vol. 21, no. 2, pp. 249–266, Aug. 2009.

[12] S. Wang and X. Yao, "Using Class Imbalance Learning for Software Defect Prediction," *IEEE Trans. Reliab.*, vol. 62, no. 2, pp. 434–443, Jun. 2013.

[13] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute Selection and Imbalanced Data: Problems in Software Defect Prediction," *2010 22nd IEEE Int. Conf. Tools with Artif. Intell.*, pp. 137–144, Oct. 2010.

[14] D. Gray, D. Bowes, N. Davey, and B. Christianson, "The misuse of the NASA Metrics Data Program data sets for automated software defect prediction," *15th Annu. Conf. Eval. Assess. Softw. Eng. (EASE 2011)*, pp. 96–103, 2011.

[15] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, Nov. 2012.

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

[17] R. S. Wahono and N. S. Herman, "Genetic Feature Selection for Software Defect Prediction," *Adv. Sci. Lett.*, vol. 20, no. 1, pp. 239–244, Jan. 2014.

[18] R. S. Wahono and N. Suryana, "Combining Particle Swarm Optimization based Feature Selection and Bagging Technique for Software Defect Prediction," *Int. J. Softw. Eng. Its Appl.*, vol. 7, no. 5, pp. 153–166, Sep. 2013.

[19] T.-Y. Wang and C.-Y. Huang, "Applying optimized BPN to a chaotic time series problem," *Expert Syst. Appl.*, vol. 32, no. 1, pp. 193–200, Jan. 2007.

[20] Y.-D. Ko, P. Moon, C. E. Kim, M.-H. Ham, J.-M. Myoung, and I. Yun, "Modeling and optimization of the growth rate for ZnO thin films using neural networks and genetic algorithms," *Expert Syst. Appl.*, vol. 36, no. 2, pp. 4061–4066, Mar. 2009.

[21] J. Lee and S. Kang, "GA based meta-modeling of BPN architecture for constrained approximate optimization," *Int. J. Solids Struct.*, vol. 44, no. 18–19, pp. 5980–5993, Sep. 2007.

[22] T.-H. (Tony) Hou, C.-H. Su, and H.-Z. Chang, "Using neural networks and immune algorithms to find the optimal parameters for an IC wire bonding process," *Expert Syst. Appl.*, vol. 34, no. 1, pp. 427–436, Jan. 2008.

[23] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, "Improving Software-Quality Predictions With Data Sampling and Boosting," *IEEE Trans. Syst. Man, Cybern. - Part A Syst. Humans*, vol. 39, no. 6, pp. 1283–1294, Nov. 2009.

[24] T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Comparing Boosting and Bagging Techniques With Noisy and Imbalanced Data," *IEEE Trans. Syst. Man, Cybern. - Part A Syst. Humans*, vol. 41, no. 3, pp. 552–568, May 2011.

[25] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[26] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Reflections on the NASA MDP data sets," *IET Softw.*, vol. 6, no. 6, p. 549, 2012.

[27] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining Third Edition*. Elsevier Inc., 2011.