Advanced Science Letters
Vol. 20, 239–244, 2014

Genetic Feature Selection for Software Defect Prediction

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

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

Recently, software defect prediction is an important research topic in the software engineering field. The accurate prediction of defect prone software modules can help the software testing effort, reduce costs, and improve the software testing process by focusing on fault-prone module. Software defect data sets have an imbalanced nature with very few defective modules compared to defect-free ones. The software defect prediction performance also decreases significantly because the dataset contains noisy attributes. In this research, we propose the combination of genetic algorithm and bagging technique for improving the performance of the software defect prediction. Genetic algorithm is applied to deal with the feature selection, and bagging technique is employed to deal with the class imbalance problem. The proposed method is evaluated using the data sets from NASA metric data repository. Results have indicated that the proposed method makes an impressive improvement in prediction performance for most classifiers.

Keywords: Software Defect Prediction, Genetic Algorithm, Feature Selection, Bagging Technique

1. INTRODUCTION

The costs of finding and correcting software defects have been the most expensive activity during both software development and software maintenance[1]. A panel at IEEE Metrics 2002[2] also concluded that manual software reviews can find only 60 percent of defects. 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.

The accurate prediction of defect-prone software modules can help direct test effort, reduce costs, improve the software testing process by focusing on fault-prone modules, and identifying refactoring candidates that are predicted as fault-prone[3]. Recent advances in software quality estimation yield building defect predictors with a mean probability of detection of 71 percent[4]. However, software fault prediction approaches are much more efficient and effective to detect software faults compared to software reviews.

Various machine learning classification algorithms have been applied for software defect prediction[5], including Logistic Regression[6], Decision Trees[7,8], Neural Networks[9,10], and Naïve-Bayes[11]. Unfortunately, software defect prediction remains a largely unsolved problem. The comparisons and benchmarking result of the defect prediction using machine learning classifiers indicate that, no significant performance differences could be detected[5] and no particular classifiers that performs the best for all the data sets[12]. There is a need of accurate defect prediction model for large-scale software system.

Two common aspects of data quality that can affect classification performance are class imbalance and noisy attributes[13] of data sets. Software defect datasets have an imbalanced nature with very few defective modules compared to defect-free ones[14]. Imbalance can lead to a model that is not practical in software defect prediction,

[1] Email Address: romi@romisatriawahono.net

 doi: 10.1166/asl.2014.5283

because most instances will be predicted as non-defect prone[15]. The software defect prediction performance also decreases significantly because the dataset contains noisy attributes[16,17]. However, the noisy data points in the datasets that cannot be confidently assumed to be erroneous using such simple method[18].

Feature selection is generally used in a machine learning field when the learning task involves high-dimensional and noisy attribute datasets. Most of the feature selection algorithms attempt to find solutions in feature selection that range between sub-optimal and near optimal regions, since they use local search throughout the entire process, instead of global search. Consequently, near-optimal to optimal solutions are quite difficult to achieve using these algorithms[19]. Genetic Algorithm can find a solution in the full search space and use a global search ability, significantly increasing the ability of finding high-quality solutions within a reasonable period of time[20].

In the current work, we propose the combination of Genetic Algorithm and Bagging technique for improving the accuracy of software defect prediction. Genetic Algorithm is applied to deal with the feature selection, and bagging technique is employed to deal with the class imbalance problem. Bagging technique is chosen due to the effectiveness in handling class imbalance[13].

2. RELATED WORKS

Feature selection is an important data preprocessing activity and has been extensively studied in the data mining and machine learning community. The main goal of feature selection is to select a subset of features that minimizes the prediction errors of classifiers. Feature selection techniques are divided into two categories: wrapper-based approach and filter-based approach. The wrapper-based approach involves training a learner during the feature selection process, while the filter-based approach uses the intrinsic characteristics of the data, based on a given metric, for feature selection and does not depend on training a learner. The primary advantage of the filter-based approach over the wrapper-based approach is that it is computationally faster. However, if computational complexity was not a factor, then a wrapper-based approach was the best overall feature selection scheme in terms of accuracy.

Once the objective in the software defect prediction is to improve the modeling quality and accuracy of software defect prediction, it has been decided to use wrapper methods. Nevertheless, wrapper methods have the associated problem of having to train a classifier for each tested feature subset. This means testing all the possible combinations of features will be virtually impossible. To solve this problem several search heuristics have been proposed, e.g. Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO). These methods are able to find fairly good solutions without searching the entire workspace.

Although feature selection has been widely applied in numerous application domains for many years, its application in the software quality prediction domain is limited[25]. Song et al.[16] applied two wrapper approaches, Forward Selection and Backward Elimination, as a feature selection for their proposed model. Song et al. concluded that a feature selection techniques, especially Forward Selection and Backward Elimination can play different roles with different learning algorithms for different data sets and that no learning scheme dominates, i.e., always outperforms the others for all data sets. This means we should choose different learning schemes for different data sets, and consequently, the evaluation and decision process is important. Wang et al.[21] applied ensemble feature selection techniques to 16 software defect data sets, and they concluded that ensembles of very few rankers are very effective and even better than ensembles of many or all rankers.

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 data set. There are two types of data sampling approaches: undersampling and oversampling[22]. Boosting is another technique which is very effective when learning from imbalanced data. Seiffert et al.[22] 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[13].

While considerable work has been done for feature selection and class imbalance problem separately, limited research can be found on investigating them both together, particularly in the software engineering field[13]. In this study, we combine Genetic Algorithm for selecting features and Bagging technique for solving the class imbalance problem, in the context of software defect prediction.

3. PROPOSED DEFECT PREDICTION METHOD

Figure1 shows an activity diagram of the integration of Bagging technique and Genetic Algorithm (GA) based feature selection. 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 features which are encoded as binary strings of 1 and 0. In this scheme, 1 represents selection of a feature and 0 means a non-selection.

As shown in Figure 1, input data set includes training data set and testing data set. Relational feature subsets are chosen and unrelated features subsets are discarded by feature subset selection. After training data set and testing data set discarded unrelated feature subsets, they become training data set of selected feature subset and testing data

set of selected feature subset. Classifiers are trained by training set with selected feature subset. Bagging[23] 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. Classification accuracy of classifier is calculated by testing set with selected feature subset. Classification accuracy of classifier, the number of selected features and the feature cost are used to construct a fitness function. Every chromosome is evaluated by the following fitness function equation.

$$fitness = W_A \times A + W_F \times \left(P + \left(\sum_{i=1}^{n_f} C_i \times F_i \right) \right)^{-1}$$

where A is classification accuracy, W_A is weight of classification accuracy, F_i is feature value, W_F is feature weight, C_i is feature cost, P is setting constant of avoiding that denominator reaches zero.

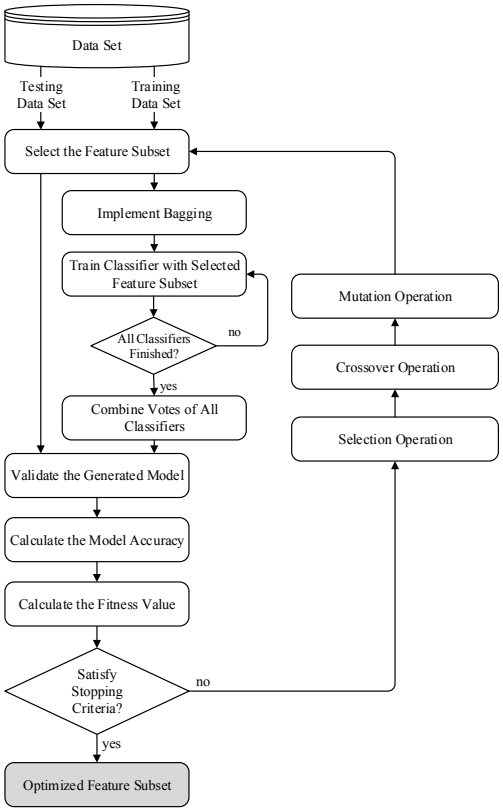

Fig.1. Activity Diagram of the Integration of Bagging Technique and Genetic Algorithm based Feature Selection

When ending condition is satisfied, the operation ends, otherwise, continue with the next generation operation. The proposed method searches for better solutions by genetic operations, including crossover, mutation and selection.

4. EXPERIMENTAL RESULT

The used platform in this experiment is 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 with Java programming language. The application software is RapidMiner 5.2.

In this research, we use nine software defect prediction data sets from NASA MDP[18]. Individual attributes per data set, together with some general statistics and descriptions, are given in Table 1. These data sets 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.

Table 1. NASA MDP Data Sets 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_cyclomatic_complexity	√		√	√	√	√	√	√	√
	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*	*77*	*37*	*37*
Number of Modules		*505*	*1571*	*458*	*127*	*403*	*1059*	*4505*	*1511*	*1347*
Number of fp Modules		*48*	*319*	*43*	*44*	*31*	*76*	*23*	*160*	*178*
Percentage of fp Modules		*9.5*	*20.31*	*9.39*	*34.65*	*7.69*	*7.18*	*0.51*	*10.59*	*13.21*

We use the state-of-the-art stratified 10-fold cross-validation for learning and testing data. 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[24].

As an accuracy indicator to evaluate the performance of classifiers in our experiments we applied *area under curve* (AUC). Lessmann et al.[5] 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.

 doi: 10.1166/asl.2014.5283

First of all, we conducted experiments on 9 NASA MDP data sets by using 10 classification algorithms. More specifically, it applies five types of classification models that include traditional statistical classifiers (Logistic Regression (LR), Linear Discriminant Analysis (LDA), and Naïve Bayes (NB)), Nearest Neighbors (k-nearest neighbor (k-NN) and K*), Neural Network (Back Propagation (BP)), Support Vector Machine (SVM), and Decision Tree (C4.5, Classification and Regression Tree (CART), and Random Forest (RF)).

The experimental results are reported in Table 2. This result confirmed Hall et al.[25] result that NB and LR, in particular, seem to be the techniques used in models that are performing relatively well in software defect prediction. Models based on Decision Tree seem to underperform due to the class imbalance. SVM techniques also perform less well, though SVM has excellent generalization ability in the situation of small sample data like NASA MDP data set.

Table 2. AUC of 10 Classifiers on 9 Data Sets (without GA and Bagging)

Classifiers		CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
Statistical Classifier	LR	0.763	0.801	0.713	0.766	0.726	0.852	0.849	0.81	0.894
	LDA	0.471	0.536	0.447	0.503	0.58	0.454	0.577	0.524	0.61
	NB	0.734	0.786	0.67	0.739	0.732	0.781	0.811	0.756	0.838
Nearest Neighbor	k-NN	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	K*	0.6	0.678	0.562	0.585	0.63	0.652	0.754	0.697	0.76
Neural Network	BP	0.713	0.791	0.647	0.71	0.625	0.784	0.918	0.79	0.883
Support Vector Machine	SVM	0.753	0.752	0.642	0.761	0.714	0.79	0.534	0.75	0.899
Decision Tree	C4.5	0.565	0.515	0.497	0.455	0.543	0.601	0.493	0.715	0.723
	CART	0.604	0.648	0.637	0.482	0.656	0.574	0.491	0.68	0.623
	RF	0.573	0.485	0.477	0.525	0.74	0.618	0.649	0.678	0.2

In the next experiment, we implemented GA and bagging technique for 10 classification algorithms on 9 NASA MDP data sets. The experimental result is shown in Table 3. The improved model for each classifier is highlighted width boldfaced print.

Table 3. AUC of 10 Classifiers on 9 Data Sets (with GA and Bagging)

Classifiers		CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
Statistical Classifier	LR	0.753	0.795	0.691	0.761	**0.742**	0.852	0.822	0.813	**0.901**
	LDA	**0.592**	**0.627**	**0.635**	**0.64**	**0.674**	**0.637**	**0.607**	**0.635**	**0.715**
	NB	0.702	**0.79**	**0.677**	0.739	0.724	**0.799**	0.805	**0.78**	**0.861**
Nearest Neighbor	k-NN	**0.666**	**0.689**	**0.67**	**0.783**	**0.656**	**0.734**	**0.554**	**0.649**	**0.732**
	K*	**0.71**	**0.822**	0.503	**0.718**	**0.68**	**0.876**	**0.877**	**0.816**	**0.893**
Neural Network	BP	**0.744**	**0.797**	**0.707**	**0.835**	**0.689**	**0.829**	0.905	**0.799**	**0.921**
Support Vector Machine	SVM	0.667	**0.767**	0.572	**0.747**	**0.659**	0.774	0.139	0.476	**0.879**
Decision Tree	C4.5	**0.64**	**0.618**	**0.658**	**0.732**	**0.695**	**0.758**	**0.642**	**0.73**	**0.844**
	CART	**0.674**	**0.818**	**0.754**	**0.709**	**0.703**	**0.819**	**0.832**	**0.842**	**0.9**
	RF	**0.706**	**0.584**	**0.605**	0.483	0.735	**0.696**	**0.901**	**0.734**	**0.601**

Figure 2 visually shows AUC comparisons of 10 algorithms on 9 NASA MDP data sets. As shown in Table 3 and Figure 2, almost all classifiers that implemented GA and bagging outperform the original method. It indicate that the integration of GA based feature selection and Bagging technique is effective to improve classification performance significantly.

Figure 2. AUC Comparisons of 9 Data Sets Classified by 10 Classifiers

Finally, in order to verify whether a significant difference between the proposed method (with GA and bagging) and a method without GA and bagging, the results of both methods are compared. We performed the statistical t-Test (*Paired Two Sample for Means*) for every classifier (algorithm) pair of without/with GA and bagging on each data set. 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 4. Although there are two classifiers (LR and NB) that have no significant difference (*P value* > 0.05), the results have indicated that those of remaining eight classifiers (LDA, k-NN, K*, BP, SVM, C4.5, CART and RF) have significant difference (*P value* < 0.05). The integration between GA and Bagging technique achieved higher classification accuracy for most classifiers. Therefore, we can conclude that the proposed

method makes an impressive improvement in prediction performance for most classifiers.

Table 4. Paired Two-tailed t-Test of without/with GA and Bagging

Classifiers		*P* value of t-Test	Result
Statistical Classifier	LR	0.156	Not Sig. (α > 0.05)
	LDA	**0.00004**	**Sig. (α < 0.05)**
	NB	0.294	Not Sig. (α > 0.05)
Nearest Neighbor	k-NN	**0.00002**	**Sig. (α < 0.05)**
	K*	**0.001**	**Sig. (α < 0.05)**
Neural Network	BP	**0.008**	**Sig. (α < 0.05)**
Support Vector Machine	SVM	**0.03**	**Sig. (α < 0.05)**
Decision Tree	C4.5	**0.0002**	**Sig. (α < 0.05)**
	CART	**0.0002**	**Sig. (α < 0.05)**
	RF	**0.01**	**Sig. (α < 0.05)**

5. CONCLUSION

Two common aspects of data quality in software defect prediction that can affect classification performance are class imbalance and noisy attributes of data sets. A novel method that integrates genetic algorithm and bagging technique for software defect prediction is proposed in this paper. Genetic algorithm is applied to deal with the noise attributes problem, and bagging technique is employed to alleviate the class imbalance problem. We conducted a comparative study of ten classifiers which is applied to nine NASA MDP data sets with context of software defect prediction. Experimental results show us that the proposed method achieved higher classification accuracy. Therefore, we can conclude that the proposed method makes an impressive improvement in prediction performance for most classifiers.

Future research will be concerned with benchmarking the proposed method with other metaheuristic optimization techniques such as bee colony or ant colony optimization, and other metalearning techniques such as boosting and sampling.

REFERENCES

[1] C. Jones. Applied Software Measurement. Mc Graw Hill, (2008).

[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. Proceedings Eighth IEEE Symposium on Software Metrics, (2002) 249-258.

[3] C. Catal. Software fault prediction: A literature review and current trends. Expert Systems with Applications, 38(4) (2011) 4626-4636.

[4] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener. Defect prediction from static code features: current results, limitations, new approaches. Automated Software Engineering, 17(4) (2010) 375-407.

[5] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transactions on Software Engineering, 34(4) (2008) 485-496

[6] G. Denaro. Estimating software fault-proneness for tuning testing activities. Proceedings of the 22nd International Conference on Software engineering (ICSE '00), (2000) 704-706.

[7] T. M. Khoshgoftaar and N. Seliya. Tree-based software quality estimation models for fault prediction. Proceedings Eighth IEEE Symposium on Software Metrics, (2002) 203-214,.

[8] T. M. Khoshgoftaar and K. Gao. Feature Selection with Imbalanced Data for Software Defect Prediction. International Conference on Machine Learning and Applications, (2009) 235-240.

[9] B.-J. Park, S.-K. Oh, and W. Pedrycz. The design of polynomial function-based neural network predictors for detection of software defects. Information Sciences, 229 (2013) 40-57.

[10] Q. Wang and B. Yu. Extract rules from software quality prediction model based on neural network. 16th IEEE International Conference on Tools with Artificial Intelligence, (2004) 191-195.

[11] T. Menzies, J. Greenwald, and A. Frank.Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Transactions on Software Engineering, 33(1) (2007) 2-13.

[12] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu. A General Software Defect-Proneness Prediction Framework. IEEE Transactions on Software Engineering, 37(3) (2011) 356-370.

[13] T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. Comparing Boosting and Bagging Techniques With Noisy and Imbalanced Data. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 41(3) (2011) 552-568.

[14] A. Tosun, A. Bener, B. Turhan, and T. Menzies. Practical considerations in deploying statistical methods for defect prediction: A case study within the Turkish telecommunications industry. Information and Software Technology, 52(11) (2010) 1242-1257.

[15] T. M. Khoshgoftaar, Y. Xiao, and K. Gao. Software quality assessment using a multi-strategy classifier. Information Sciences, (2010) .

[16] T. Wang, W. Li, H. Shi, and Z. Liu. Software Defect Prediction Based on Classifiers Ensemble. Journal of Information & Computational Science. 16 (8) (2011) 4241-4254.

[17] S. Kim, H. Zhang, R. Wu, and L. Gong. Dealing with noise in defect prediction. Proceeding of the 33rd International Conference on Software Engineering, (2011) 481-490.

[18] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, Reflections on the NASA MDP data sets. IET Software, 6(6) (2012) 549-558.

[19] M. M. Kabir, M. Shahjahan, and K. Murase. A new hybrid ant colony optimization algorithm for feature selection. Expert Systems with Applications, 39(3) (2012) 3747-3763.

[20] S. C. Yusta. Different metaheuristic strategies to solve the feature selection problem. Pattern Recognition Letters, 30(5) (2009) 525-534.

[21] H. Wang, T. M. Khoshgoftaar, and A. Napolitano. Software measurement data reduction using ensemble techniques. Neurocomputing, 92 (2012) 124-132.

[22] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse. Improving Software-Quality Predictions With Data Sampling and Boosting. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 39(6) (2009) 1283-1294.

[23] L. Breiman. Bagging predictors. Machine Learning, 24(2) (1996) 123-140.

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

[25] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. IEEE Transactions on Software Engineering, 38(6) (2012) 1276-1304.