International Journal of Software Engineering and Its Applications
Vol.7, No.5 (2013), pp.153-166
http://dx.doi.org/10.14257/ijseia.2013.7.5.16

Combining Particle Swarm Optimization based Feature Selection and Bagging Technique for Software Defect Prediction

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

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

Abstract

The costs of finding and correcting software defects have been the most expensive activity in software development. 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. Recently, static code attributes are used as defect predictors in software defect prediction research, since they are useful, generalizable, easy to use, and widely used. However, two common aspects of data quality that can affect performance of software defect prediction are class imbalance and noisy attributes. In this research, we propose the combination of particle swarm optimization and bagging technique for improving the accuracy of the software defect prediction. Particle swarm optimization 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, machine learning, particle swarm optimization, feature selection, bagging*

1. Introduction

A software defect is an error, failure, or fault in a computer program or system that produces an incorrect or unexpected result, and decreased the quality of the software. The costs of finding and correcting software defects have been the most expensive activity during both software development and software maintenance. Software defect increases in cost over the software development phase. During the coding phase, finding and correcting defects costs $977 per defect. In the system-testing phase, the cost increases to $7,136 per defect. If a defect survives to the maintenance phase, then the cost to find and remove increases to $14,102 [1].

A panel at IEEE Metrics 2002 [2] concluded that manual software reviews can find only 60 percent of defects. Recent studies show that the probability of detection of fault prediction models may be higher than probability of detection of software reviews. Menzies et al. found defect predictors with a probability of detection of 71 percent [3]. This is higher than other currently used industrial methods such as manual code 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 [4], and finally improve the quality of software [5]. Therefore,

ISSN: 1738-9984 IJSEIA

software fault prediction approaches are much more cost-effective to detect software faults compared to software reviews.

Machine learning classification algorithm is a popular approach for software defect prediction [6]. It categorizes the software code attributes into defective or not defective, which is completed by means of a classification model derived from software metrics data of previous development projects [7]. 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 independent variables and fault data is used as the dependent variable [4]. Variable predictors of the prediction model are computed by using previous software metrics and fault data.

Various types of classification algorithms have been applied for software defect prediction, including logistic regression [8], decision trees [9], neural networks [10], 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 [6] 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, because most instances will be predicted as non-defect prone [15]. Learning from imbalanced datasets is difficult. The insufficient information that is associated with the minority class impedes making a clear understanding of the inherent structure of the dataset [16]. The software defect prediction performance also decreases significantly because the dataset contains noisy attributes [17] [18]. However, the noisy data points in the datasets that cannot be confidently assumed to be erroneous using such simple method [19].

Feature selection is generally used in machine learning when the learning task involves high-dimensional and noisy attribute datasets. Most of the feature selection algorithms, 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. On the other hand, these search algorithms utilize a partial search over the feature space, and suffer from computational complexity. Consequently, near-optimal to optimal solutions are quiet difficult to achieve using these algorithms [20].

Metaheuristic optimization 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 [21]. Mostly used metaheuristic optimization for feature selection include genetic algorithm (GA), particle swarm optimization (PSO) and ant colony optimization (ACO).

In the current work, we propose the combination of particle swarm optimization (PSO) and bagging technique for improving the accuracy of software defect prediction. Particle swarm optimization 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]. The proposed method is evaluated using the state-of-the-art and public datasets from NASA metric data repository.

2. Related Work

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 this work 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. Most of the feature selection strategies attempt to find solutions that range between sub-optimal and near optimal regions, since they use local search throughout the entire process, instead of global search. On the other hand, these search algorithms utilize a partial search over the feature space, and suffer from computational complexity. Consequently, near-optimal to optimal solutions are quiet difficult to achieve using these algorithms. As a result, many research studies now focus on metaheuristic optimization techniques [20]. The significance of metaheuristic optimization techniques is that they can find a solution in the full search space on the basis of activities of multi-agent systems that use a global search ability utilizing local search appropriately, thus significantly increasing the ability of finding very high-quality solutions within a reasonable period of time. Metaheuristic optimization techniques have been developed in several domains and include algorithms like simulated annealing, tabu-search, as well as bio-inspired methods like genetic algorithms, evolution strategies, ant colony optimization and particle swarm optimization. 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. Song *et al.*, [12] 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.*, [22] 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 [23]. Boosting is another technique which is very effective when learning from imbalanced data. Compared to data sampling, boosting has received

relatively little attention in data-mining research with respect to class imbalance. However, Seiffert *et al.*, [23] show that boosting performs very well. Bagging, may outperform boosting when data contain noise [24], because boosting may attempt to build models to correctly classify noisy examples. In this study we apply bagging technique, because Khoshgoftaar *et al.*, [13] showed that the bagging techniques generally outperform boosting, and hence in noisy data environments. Therefore bagging is the preferred method for handling class imbalance.

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 particle swarm optimization for selecting features and bagging technique for solving the class imbalance problem, in the context of software defect prediction.

3. Proposed Defect Prediction Method

3.1. Particle Swarm Optimization

Particle swarm optimization (PSO) is an emerging population-based meta-heuristic that simulates social behavior such as birds flocking to a promising position to achieve precise objectives in a multi-dimensional space. PSO performs searches using a population (*swarm*) of individuals (*particles*) that are updated from iteration to iteration. The size of population is denoted as p_{size}. To discover the optimal solution, each particle changes its searching direction according to two factors, its own best previous experience (*pbest*) and the best experience of all other members (*gbest*). Shi and Eberhart [25] called *pbest* the cognition part, and *gbest* the social part.

Each particle represents a candidate position (solution). A particle is considered as a point in a *D*-dimension space, and its status is characterized according to its position and velocity. The *D*-dimensional position for the particle *i* at iteration *t* can be represented as $x_i^t = \{x_{i1}^t, x_{i2}^t, \dots, x_{iD}^t\}$. Likewise, the velocity (distance charge) for particle *i* at iteration *t*, which is also a *D*-dimension vector, can be described as $v_i^t = \{v_{i1}^t, v_{i2}^t, \dots, v_{iD}^t\}$.

In the later version of PSO, a new parameter, called inertia weight introduced by [25] due to control over the previous velocity of the particles. Let $p_i^t = \{p_{i1}^t, p_{i2}^t, \dots, p_{iD}^t\}$ represent the best solution that particle *i* has obtained until iteration *t*, and $p_g^t = \{p_{g1}^t, p_{g2}^t, \dots, p_{gD}^t\}$ denote the best solution obtained from p_i^t in the population at iteration *t*. To search for the optimal solution, each particle changes its velocity according to the cognitive and social part using equation (1).

$$V_{id}^t = w * V_{id}^{t-1} + c_1 r_1 \left(P_{id}^t - x_{id}^t\right) + c_2 r_2 \left(P_{gd}^t - x_{id}^t\right), \quad d = 1, 2, \dots D \tag{1}$$

Note that, c_1 indicates the cognitive learning factor, c_2 indicates the social learning factor, inertia weight (*w*) is used to slowly reduce the velocity of the particles to keep the swarm under control, and r_1 and r_2 are random numbers uniformly distributed in *U(0,1)*.

Each particle then moves to a new potential solution based on the equation (2).

$$X_{id}^{t+1} = X_{id}^t + V_{id}^t, \quad d = 1, 2, \dots D \tag{2}$$

The basic process of the PSO algorithm is given as follows:

1. *Initialization*: Randomly generate initial particles

2. *Fitness*: Measure the fitness of each particle in the population
3. *Update*: Compute the velocity of each particle with equation (1)
4. *Construction*: For each particle, move to the next position according to equation (2)
5. *Termination*: Stop the algorithm if termination criterion is satisfied, and return to step 2 (Fitness) otherwise

The iteration is terminated if the number of iteration reaches the pre-determined maximum number of iteration.

3.2. Bagging Technique

Bagging (Bootstrap Aggregating) was proposed by Leo Breiman in 1994 [26] 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. It also reduces variance and helps to avoid overfitting.

Description of the bagging technique is as follows. Given a standard training set D of size n, bagging generates m new training sets D_i, each of size $n'<n$, by sampling from D uniformly and with replacement. By sampling with replacement, some observations may be repeated in each D_i. If $n'=n$, then for large n the set D_i is expected to have the fraction $(1 - 1/e)$ of the unique examples of D, the rest being duplicates. This kind of sample is known as a bootstrap sample. The m models are fitted using the above m bootstrap samples and combined by averaging the output (for regression) or voting (for classification).

Bagging leads to improvements for unstable procedures [26], which include neural network, classification and regression trees, and subset selection in linear regression. On the other hand, it can mildly degrade the performance of stable methods such as K-nearest neighbors.

3.3. Proposed Method

Figure 1 shows an activity diagram of the integration of bagging technique and particle swarm optimization based feature selection. A group of particles are random generated, dimensional discrete binary variable. The particle length is the total characteristics number, and if and each particle is one the first *i*-bit is 1, then the first feature *i* was selected, otherwise it will be shielded. Each particle represents a feature subset, which is a candidate solution. Implement bagging technique and train the classifier on the larger training set based on the selected feature subset and the type of kernel. If all classifiers are finished, combine votes of all classifiers. Finally, measure validation accuracy on testing data set via the generated model.

We use the accuracy of classifier to evaluate fitness size, the higher accuracy rate, the greater the fitness. the goal of select the feature characteristic subset that is to the achieve the use a small number of same or better classification results, so the fitness function evaluation should also take into consideration the number of characteristics, given same accuracy to two characteristics of a subset, the one which have fewer characteristic number will be higher fitness.

Update the global and personal best according to the fitness evaluation results. Record the average validation accuracy for the global and personal best. If the fitness is better than the particle's best fitness, then the position vector is saved for the particle. If the particle's fitness is better than the global best fitness, then the position vector is saved for the global best. Finally the particle's velocity and position are updated until the termination condition is

satisfied. To avoid overtraining, we observe the validation accuracy curve, and stop training when the iteration has the best validation accuracy during the training process. With the stopping training iteration determined in the previous step, recall the recorded feature subset and the type of kernel in the stopping iteration.

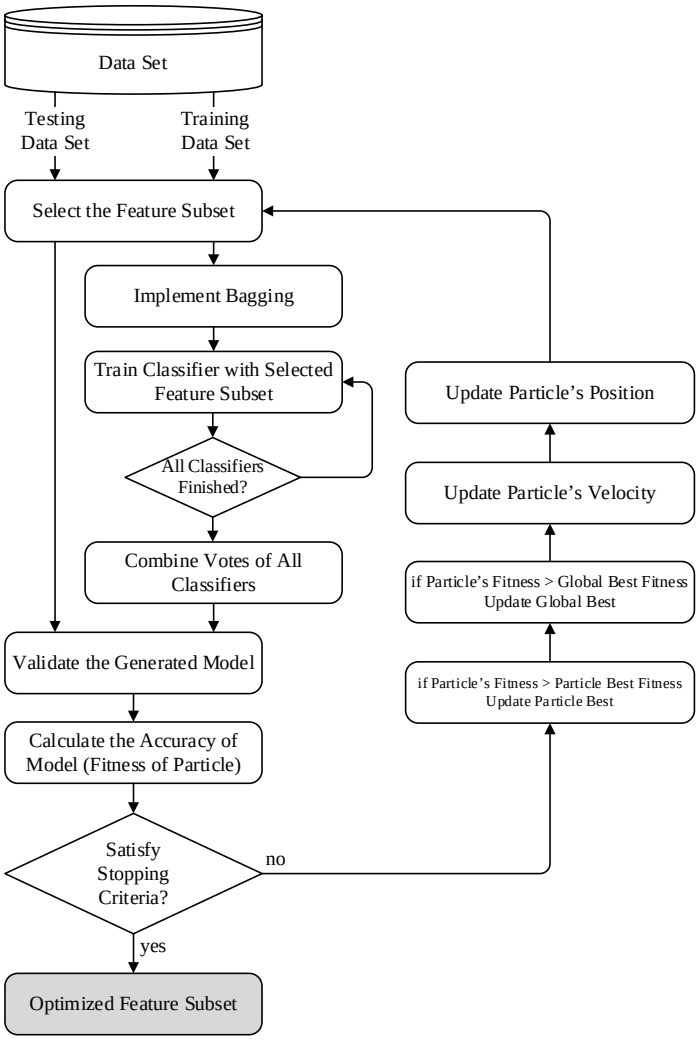

Figure 1. Activity Diagram of the Integration of Bagging Technique and Particle Swarm Optimization based Feature Selection

4. Experiments

The framework of our experiment is shown in Figure 2. The framework is comprised 1) a data sets 2) a feature selection, 3) a meta-learning, 4) a learning algorithm, 5) a model validation, 6) a model evaluation and 7) a model comparison. The used platform 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.

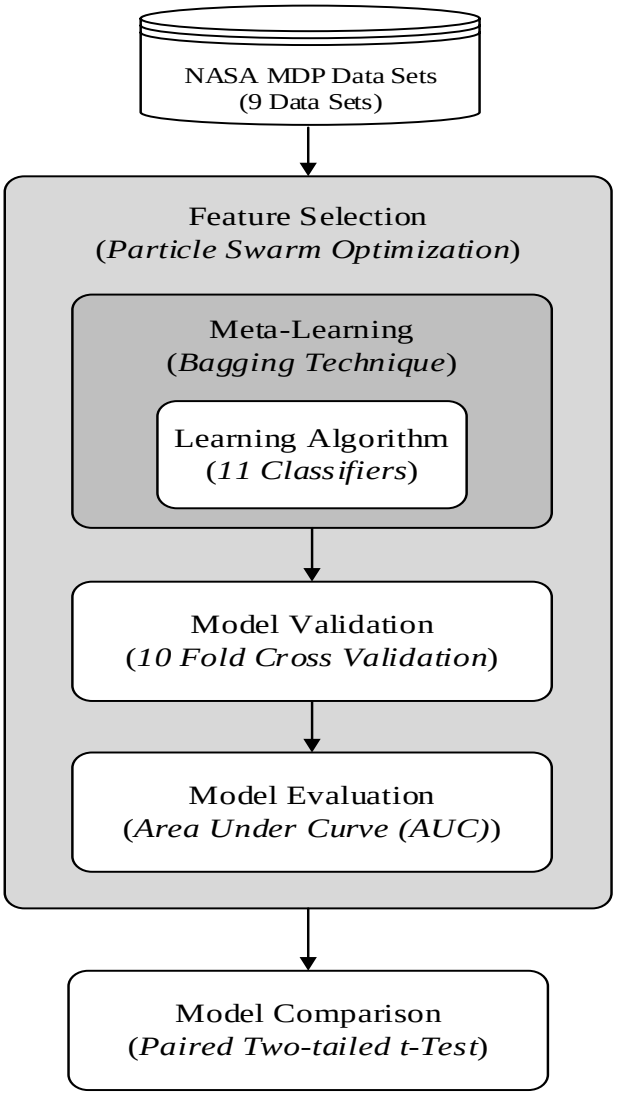

Figure 2. The Framework of Experiment

4.1. Data Sets

One of the most important problems for software fault prediction studies is the usage of nonpublic (private) data sets. Several companies developed fault prediction models using proprietary data and presented these models in conferences. However, it is not possible to compare results of such studies with results of our own models because their datasets cannot be reached. The use of public datasets makes our research repeatable, refutable, and verifiable [27]. Recently, state-of-the-art public data sets used for software defect prediction research is available in NASA Metrics Data (MDP) repository [19].

The data used in this research are collected from the NASA MDP repository. NASA MDP repository is a database that stores problem, product, and metrics data [19]. The primary goal of this data repository is to provide project data to the software community. In doing so, the Metrics Data Program collects artifacts from a large NASA dataset, generates metrics on the artifacts, and then generates reports that are made available to the public at no cost. The data

that are made available to general users have been sanitized and authorized for publication through the Metrics Data Program Web site by officials representing the projects from which the data originated.

Each NASA data set is comprised of several software modules, together with their number of faults and characteristic code attributes. After preprocessing, modules that contain one or more errors were labeled as fault-prone, whereas error-free modules were categorized as not-fault-prone. Besides line of codes (LOC) counts, the NASA MDP data sets include several Halstead attributes as well as McCabe complexity measures. The former estimates reading complexity by counting operators and operands in a module, whereas the latter is derived from a module's flow graph.

Some researchers endorse the static code attributes defined by McCabe and Halstead as defect predictors in the software defect prediction. McCabe and Halstead are module-based metrics, where a module is the smallest unit of functionality. Static code attributes are used as defect predictors, since they are useful, generalizable, easy to use, and widely used [14].

In this research, we use nine software defect prediction data sets from NASA MDP. Individual attributes per data set, together with some general statistics and descriptions, are given in Table 2. These data sets have various scales of 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.

4.2. Model Validation

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 (Table 1). Each time, we chose another part of dataset for testing and used the remaining nine parts for learning. After, we calculated the average values and the deviation values from the ten different testing results. 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 [28].

Table 1. Stratified 10 Fold Cross Validation

n-validation	Dataset's Partition									
1	■									
2		■								
3			■							
4				■						
5					■					
6						■				
7							■			
8								■		
9									■	
10										■

Table 2. 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*

4.3. Model Evaluation

We applied *area under curve* (AUC) as an accuracy indicator in our experiments to evaluate the performance of classifiers. AUC is area under ROC curve. Lessmann *et al.*, [6] 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. Furthermore, the AUC has a clear statistical interpretation. It measures the probability that a classifier ranks a randomly chosen fault-prone module higher than a randomly chosen non-fault-prone module. Consequently, any classifier achieving AUC well above 0.5 is demonstrably effective for identifying fault-prone modules and gives valuable advice as to which modules should receive particular attention in software testing.

A roughs guide for classifying the accuracy of a diagnostic test using AUC is the traditional system, presented below [29]:

- 0.90 - 1.00 = excellent classification
- 0.80 - 0.90 = good classification

- 0.70 - 0.80 = fair classification
- 0.60 - 0.70 = poor classification
- 0.50 - 0.60 = failure

4.4. Model Comparison using Paired Two-tailed t-Test

A paired t-test compares two samples in cases where each value in one sample has a natural partner in the other. A paired t-test looks at the difference between paired values in two samples, takes into account the variation of values within each sample, and produces a single number known as a t-value. In this research, we have applied pair wise t-tests over mean values of each datasets in order to determine statistical significance of results with $\propto = 0.05$.

5. Result and Analysis

First, we conducted experiments on 9 NASA MDP data sets by using 11 classification algorithms. The experimental results are reported in Table 3. This result confirmed Hall et al. [5] result that Naive Bayes and Logistic Regression, 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. Support Vector Machine (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 3. AUC of 11 Classifiers on 9 Data Sets (without PSO and Bagging)

Classifiers		CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
Statistical Classifier	Logistic Regression	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
	Naive Bayes	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	Back Propagation	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
	LibSVM	0.692	0.751	0.596	0.54	0.628	0.674	0.907	0.742	0.699
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
	Random Forest	0.573	0.485	0.477	0.525	0.74	0.618	0.649	0.678	0.2

In the next experiment, we implemented PSO and bagging technique for 11 classification algorithms on 9 NASA MDP data sets. The experimental result is shown in Table 4. The improved model for each classifier is highlighted width boldfaced print. As shown in Table 4, almost all classifiers that implemented PSO and bagging outperform the original method. It indicate that the integration of PSO based feature selection and bagging technique is effective to improve classification performance. Support Vector Machine (SVM) techniques still perform less well and no significant improvement. It indicates that feature selection and bagging are not the answer of the SVM performance problem. This result also confirmed that SVM may be underperforming as they require parameter optimization for best performance

[5]. The integration between PSO and bagging technique affected significantly on the performance of the class imbalance suffered classifiers, such as C4.5 and CART.

Table 4. AUC of 11 Classifiers on 9 Data Sets (with PSO and Bagging)

Classifiers		CM1	KC1	KC3	MC2	MW1	PC1	PC2	PC3	PC4
Statistical Classifier	Logistic Regression	0.738	0.798	0.695	**0.78**	**0.751**	0.848	0.827	**0.816**	**0.897**
	LDA	0.469	**0.627**	**0.653**	**0.686**	**0.632**	**0.665**	0.571	**0.604**	**0.715**
	Naive Bayes	**0.756**	**0.847**	**0.71**	0.732	**0.748**	**0.79**	**0.818**	**0.78**	**0.85**
Nearest Neighbor	k-NN	**0.632**	**0.675**	**0.578**	**0.606**	**0.648**	**0.547**	**0.594**	**0.679**	**0.738**
	K*	**0.681**	**0.792**	**0.66**	**0.725**	0.572	**0.822**	0.814	**0.809**	**0.878**
Neural Network	Back Propagation	0.7	**0.799**	**0.726**	**0.734**	**0.722**	**0.809**	0.89	**0.823**	**0.915**
Support Vector Machine	SVM	0.721	0.723	**0.67**	0.756	0.667	**0.792**	0.294	0.735	**0.903**
	LibSVM	**0.701**	0.721	**0.657**	0.507	**0.665**	**0.732**	0.839	**0.744**	**0.87**
Decision Tree	C4.5	**0.682**	**0.606**	**0.592**	**0.648**	**0.615**	**0.732**	**0.732**	**0.78**	**0.769**
	CART	**0.611**	**0.679**	**0.787**	**0.679**	**0.682**	**0.831**	**0.794**	**0.845**	**0.912**
	Random Forest	**0.62**	**0.604**	**0.557**	**0.533**	0.714	**0.686**	**0.899**	**0.759**	**0.558**

Finally, in order to verify whether a significant difference between the proposed method (with PSO and bagging) and a method without PSO and bagging, the results of both methods are compared. We performed t-Test (*Paired Two Sample for Means*) for every classifier (algorithm) pair of without/with PSO and bagging on each data set. We set significance level (α) to be 0.05. The result is shown in Table 5. Although there are three classifiers that have no significant difference ($P > 0.05$), the results have indicated that those of remaining eight classifiers have significant difference ($P < 0.05$). Therefore, we can conclude that the proposed method (the integration between PSO and bagging technique) makes an impressive improvement in prediction performance for most classifiers.

Table 5. Paired Two-tailed t-Test of without/with PSO and Bagging

Classifiers		*P* value of t-Test	Result
Statistical Classifier	Logistic Regression	0.323	Not Sig. ($P > 0.05$)
	LDA	0.003	**Sig. ($P < 0.05$)**
	Naive Bayes	0.007	**Sig. ($P < 0.05$)**
Nearest Neighbor	k-NN	0.00007	**Sig. ($P < 0.05$)**
	K*	0.001	**Sig. ($P < 0.05$)**
Neural Network	Back Propagation	0.03	**Sig. ($P < 0.05$)**
Support Vector Machine	SVM	0.09	Not Sig. ($P > 0.05$)
	LibSVM	0.178	Not Sig. ($P > 0.05$)
Decision Tree	C4.5	0.0002	**Sig. ($P < 0.05$)**
	CART	0.002	**Sig. ($P < 0.05$)**
	Random Forest	0.01	**Sig. ($P < 0.05$)**

6. Conclusion

A novel method that integrate particle swarm optimization and bagging technique is proposed in this paper, to improve the accuracy of software defect prediction. Particle swarm optimization 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 eleven classifiers which is applied to nine NASA MDP data sets with context of software defect prediction. Experimental results show us 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 genetic algorithm or ant colony optimization, and other metalearning techniques, such as boosting and sampling. The investigation of more sophisticated metaheuristic optimization techniques for optimizing parameter of SVM for software defect prediction will be studied in our future work.

References

[1] B. Boehm and V. R. Basili, "Top 10 list [software development]", Computer, vol. 34, no. 1, **(2001)**, pp. 135-137.
[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, **(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", Automated Software Engineering, vol. 17, no. 4, **(2010)**, pp. 375-407.
[4] C. Catal, "Software fault prediction: A literature review and current trends," Expert Systems with Applications, vol. 38, no. 4, **(2011)**, pp. 4626-4636.
[5] 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, vol. 38, no. 6, **(2012)**, pp. 1276-1304.
[6] 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, vol. 34, no. 4, **(2008)**, pp. 485-496.
[7] N. Gayatri, S. Nickolas, A. Reddy, S. Reddy and A. V. Nickolas, "Feature Selection Using Decision Tree Induction in Class level Metrics Dataset for Software Defect Predictions", Proceedings of the World Congress on Engineering and Computer Science, vol. I, **(2010)**, pp. 124-129.
[8] G. Denaro, "Estimating software fault-proneness for tuning testing activities", Proceedings of the 22nd International Conference on Software engineering - ICSE '00, **(2000)**, pp. 704-706.
[9] T. M. Khoshgoftaar and K. Gao, "Feature Selection with Imbalanced Data for Software Defect Prediction", 2009 International Conference on Machine Learning and Applications, **(2009)**, pp. 235-240.
[10] 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, **(2011)**.
[11] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors", IEEE Transactions on Software Engineering, vol. 33, no. 1, **(2007)**, pp. 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, vol. 37, no. 3, **(2011)**, pp. 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, vol. 41, no. 3, **(2011)**, pp. 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, vol. 52, no. 11, **(2010)**, pp. 1242-1257.
[15] T. M. Khoshgoftaar, Y. Xiao and K. Gao, "Software quality assessment using a multi-strategy classifier", Information Sciences, **(2010)**.
[16] Y. Liu, X. Yu, J. X. Huang and A. An, "Combining integrated sampling with SVM ensembles for learning from imbalanced datasets", Information Processing & Management, vol. 47, no. 4, **(2011)**, pp. 617-631.

[17] T. Wang, W. Li, H. Shi and Z. Liu, "Software Defect Prediction Based on Classifiers Ensemble", Journal of Information & Computational Science 8:, vol. 16, **(2011)** December, pp. 4241-4254.
[18] S. Kim, H. Zhang, R. Wu and L. Gong, "Dealing with noise in defect prediction", Proceeding of the 33rd International Conference on Software engineering - ICSE '11, **(2011)**, pp. 481-490.
[19] D. Gray, D. Bowes, N. Davey, Y. Sun and B. Christianson, "Reflections on the NASA MDP data sets", IET Software, vol. 6, no. 6, **(2012)**, pp. 549.
[20] M. M. Kabir, M. Shahjahan and K. Murase, "A new hybrid ant colony optimization algorithm for feature selection", Expert Systems with Applications, vol. 39, no. 3, **(2012)**, pp. 3747-3763.
[21] S. C. Yusta, "Different metaheuristic strategies to solve the feature selection problem", Pattern Recognition Letters, vol. 30, no. 5, **(2009)**, pp. 525-534.
[22] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "Software measurement data reduction using ensemble techniques", Neurocomputing, vol. 92, **(2012)**, pp. 124-132.
[23] 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, vol. 39, no. 6, **(2009)**, pp. 1283-1294.
[24] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques", Software: Practice and Experience, vol. 41, no. 5, **(2011)**, pp. 579-606.
[25] Y. Shi and R. Eberhart, "A modified particle swarm optimizer", 1998 IEEE International Conference on Evolutionary Computation Proceedings IEEE World Congress on Computational Intelligence Cat No98TH8360, vol. 189, no. 2, **(1998)**, pp. 69-73.
[26] L. Breiman, "Bagging predictors", Machine Learning, vol. 24, no. 2, **(1996)**, pp. 123-140.
[27] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem", Information Sciences, vol. 179, no. 8, **(2009)**, pp. 1040-1058.
[28] I. H. Witten, E. Frank and M. A. Hall, Data Mining Third Edition. Elsevier Inc., **(2011)**.
[29] F. Gorunescu, "Data Mining: Concepts, Models and Techniques", Springer-Verlag Berlin Heidelberg, vol. 12, **(2011)**.

Authors

Romi Satria Wahono received B.Eng and M.Eng degrees in Computer Science in 1999 and 2001, respectively from Saitama University, Japan. Currently working towards a PhD with Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka. He is a lecturer at the Graduate School of Computer Science, Dian Nuswantoro University, Indonesia. He is also a founder and chief executive officer of a software development company in Indonesia. His current research interests include Software Engineering and Machine Learning. Professional member of the ACM and IEEE Computer Society.

Nanna Suryana received his B.Sc. in Soil and Water Eng. (Bandung, Indonesia), M.Sc. in Comp. Assisted for Geoinformatics and Earth Science, (Enschede, Holland), Ph.D. in Geographic Information System (GIS) (Wageningen, Holland). He is currently holding a position of Director of International Office and professor at Faculty of Information Technology and Communication (FTMK) of Universiti Teknikal Malaysia Melaka (UTEM). His current research interest is in field of GIS and Data Mining.