Expert Systems with Applications 42 (2015) 1872–1879

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A comparison of some soft computing methods for software fault prediction

Ezgi Erturk [a], Ebru Akcapinar Sezer [b,*]

[a] The Scientific and Technological Research Council of Turkey (TUBITAK), Software Technologies Research Institute, Ankara, Turkey
[b] Hacettepe University, Department of Computer Engineering, 06800 Ankara, Turkey

ARTICLE INFO

Article history:
Available online 23 October 2014

Keywords:
Software fault prediction
McCabe metrics
Adaptive neuro fuzzy systems
Artificial Neural Networks
Support Vector Machines

ABSTRACT

The main expectation from reliable software is the minimization of the number of failures that occur when the program runs. Determining whether software modules are prone to fault is important because doing so assists in identifying modules that require refactoring or detailed testing. Software fault prediction is a discipline that predicts the fault proneness of future modules by using essential prediction metrics and historical fault data. This study presents the first application of the Adaptive Neuro Fuzzy Inference System (ANFIS) for the software fault prediction problem. Moreover, Artificial Neural Network (ANN) and Support Vector Machine (SVM) methods, which were experienced previously, are built to discuss the performance of ANFIS. Data used in this study are collected from the PROMISE Software Engineering Repository, and McCabe metrics are selected because they comprehensively address the programming effort. ROC-AUC is used as a performance measure. The results achieved were 0.7795, 0.8685, and 0.8573 for the SVM, ANN and ANFIS methods, respectively.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

Software quality has experienced an increase in demand in the past ten years because of the significant role that software systems play in daily human life. Tolerance of unpredictable software behavior during run time is decreasing because of the high cost of this type of behavior and because it may cause irrecoverable situations. As a result, studies on software quality attempt to increase the quality of the software development life cycle and produce high quality software systems. The major expectation from high quality software is its reliability; therefore, the number of failures that occur when the program is running must be minimized to ensure reliable software. The main causes of the failures are faults generated by errors in human action. Errors, failures, faults and defects are defined as follows. An error is a missing human action or a human action that embeds particular mistakes into the product. For example, mistakes made during code implementation are accepted as errors. Failure occurs when the behavior of the software does not meet user expectations. A fault is an incorrect step, code, process, data definition or physical defect that occurs in hardware or software. A fault is the primary cause of failures in a software program. For example, the software can go into an unexpected state because of a programmer mistake and, as a result, this type of fault may cause a failure. The term defect is used as a general expression of error, failure and fault. In summary, errors may cause faults and faults may cause failures when the software program runs, and the term defect includes all of these terms.

Determination of fault-prone software modules is an important process because it helps to identify modules that require refactoring or detailed testing. In this way, more qualified software products may be developed. Software fault prediction is a discipline that predicts the fault proneness of future modules using essential prediction metrics and historical fault data. By considering software fault prediction systems, a project schedule can be planned more efficiently, especially for testing and maintenance phases. The benefits of software fault prediction are listed below (Catal, 2011).

- The test process can be refined, thus increasing system quality.
- Specifying modules that require refactoring during the maintenance phase is possible.
- Applying software fault prediction using class-level metrics during the design phase enables selecting the best of the design alternatives.
- Software fault prediction provides stability and high assurance of the software system.

* Corresponding author. Tel.: +90 3122977500; fax: +90 3122977502.
E-mail addresses: erturkezgi@gmail.com (E. Erturk), ebruakcapinarsezer@gmail.com (E.A. Sezer).

http://dx.doi.org/10.1016/j.eswa.2014.10.025
0957-4174/© 2014 Elsevier Ltd. All rights reserved.

- Software fault prediction can reduce the time and effort spent in the code review process.

Because of these benefits, predicting software faults becomes an important research problem. In fact, the aim of prediction is valid and available in many different areas, and many proposed prediction or classification methods exist: decision trees, neural networks, vector machines, logistic regression, etc. Moreover, some of these methods are applied to the software fault prediction problem listed in Section 2. This study selects three different soft computing methods—Adaptive Neuro Fuzzy Inference System (ANFIS), Support Vector Machine (SVM), and Artificial Neural Network (ANN)—to build software fault prediction models and compares their performance results. In fact, previous studies have applied ANN and SVM to this problem (e.g. Thwin and Quah (2003), Gondra (2008), Mahaweerawat, Sophatsathit, Lursinsap, and Musílek (2004), Malhotra (2014), Xing, Guo, and Lyu (2005)). However, ANFIS has not been applied to this problem using a comparative approach.

ANFIS is a powerful prediction method that combines learning ability and expert knowledge and achieves many successful results in predicting problems in different areas (e.g. Daoming and Jie (2006), Lo (2003), Najah, El-Shafie, Karim, and Jaafar (2012), Perez, Gonzalez, and Dopico (2010), Pradhan, Sezer, Gokceoglu, and Buchroithner (2010), Sezer, Pradhan, and Gokceoglu (2011)). As emphasized in Catal and Diri (2009) selected algorithm for identification of the faulty software modules is important as much as the selected parameters. At this point we contribute the solution methods of this problem by proposing ANFIS. The difference between ANFIS and the other data driven methods (ANN, SVM, and decision tree (DT)) is that ANFIS uses expert knowledge to build the model and uses data to optimize it. Actually, modeler has to decide which parameters are used regardless of the employed method when he/she uses pure data driven methods. However, modeler also has to specify the membership function type and number, or fuzzy sets for each parameter while modeling with ANFIS and this information directly specify the way of fuzziness or vagueness handling in the predictive model. For example, if "line of code (loc)" parameter is used while modeling with ANN, its data format and its relationship between the output parameter are considered. However, if the predictive method is ANFIS, in addition to these considerations, expert should specify the number of fuzzy sets which will be used for "loc" parameter. If it is specified as 2 (i.e. low, high) with triangular membership function, it means that vagueness of the model specified at the maximum level. If 3 fuzzy sets (i.e. low, moderate, high) are used in the form of triangular membership function, vagueness of the model decreases according to previous case. The decisions of the expert are based on the natural distribution of the values of parameter, the natural class numbers of the parameter, degree of fuzziness while passing one class to other and the relationship between the classes of input and output parameters. Consequently, it is possible to say that modeling with ANFIS requires more awareness about the conditions and results embedded in the data. Thus, use of each parameter is decided one by one, not as a whole parameter set. As a result, use of ANFIS becomes important because of three reasons (i) ANFIS is a powerful predictive method but, it has not been experienced before for software fault prediction problem (ii) modeling with ANFIS triggers detailed reconsideration of the each input parameter even if it has been commonly used in previous studies (iii) properties of ANFIS model gives some valuable information (best class number for each parameter, vagueness degree, distribution of the data, most effective rules) to the modeler of fully expert based systems. In other words, ANFIS is easier way of experimentation for an expert than other machine learning methods to optimize his/her knowledge.

Based on these reasons, the ANFIS is employed to predict software faultiness for the first time in this study and also two of commonly used machine learning methods (ANN and SVM) are employed to make comparison between ANFIS's and their performances. This study uses McCabe metrics (Thomas, 1976) as software features and the dataset of the experiments is created by composing projects using the PROMISE Software Engineering Repository (Sayyad & Menzies (2005); Promise Software Engineering Repository Public Datasets (2013)). In experimentation step, all McCabe metrics (loc, v(g), ev(g), and iv(g)) are used in models and performance of them are evaluated at first. After examination of the faulty cases, experiments which use 3 McCabe metrics (loc, v(g), and iv(g)) are organized at second. The models' performances are compared using area under ROC curve (AUC). The performance results show that ANFIS is a competitive and strong method to solve this problem and encourages us to conduct further studies. Additionally, use of 3 McCabe metrics is proposed instead of all them. Experimental results show SVM and ANFIS achieve better results with 3 parameters than 4.

2. Related works

The studies relating with the software fault prediction problem are summarized here into two parts such as older studies published before the year of 2013 and the studies presented in last two years. Catal (2011) surveyed 90 papers on software fault prediction that were published between 1990 and 2009. The most important contribution of the study was that it provides a guide for researchers on software metrics, methods used for software fault prediction, datasets, and performance evaluation criteria. Catal, Sevim, and Diri (2011) developed an Eclipse-based software fault prediction tool using machine learning and statistical techniques. The objectives of the study were to both practically and theoretically predict faults in software programs. They preferred the Naive Bayes algorithm for its high performance. Catal and Diri (2009) investigated the effect of dataset size, metrics sets and feature selection techniques on software fault prediction problems, which were previously not researched in this research area. They used the Random Forest algorithm and Artificial Immune Systems as machine learning methods and the PROMISE repository as a dataset. As a result, they determined that the selected algorithm is more important than the selected metrics. This finding strongly supports employing the ANFIS method for this problem. Mahaweerawat, Sophatsathit, Lursinsap, and Musílek (2006) developed an enhanced model named MASP to predict and identify faults in object-oriented software systems. The MASP model can filter appropriate metrics for particular fault types. The aim of the Vandecruys et al. (2008) study was to efficiently predict faulty software modules and support software development by investigating software repositories using data mining techniques (ant colony optimization-based AntMiner+). Alsmadi and Najadat (2011) aimed to predict fault-prone modules and to determine relationships among them. They considered that attributes with high correlations between them could negatively affect each other. Hu, Xie, Ng, and Levitin (2007) proposed correcting faults in addition to predicting faulty parts of software. For this purpose, they iteratively applied neural networks and a genetic algorithm. The genetic algorithm increased the performance of the prediction model. Gondra (2008) attempted to show which software metric was more important for fault prediction and used ANN and SVM methods for this purpose. Furthermore, he compared the results of the models built using ANN and SVM. Zimmermann, Premraj, and Zeller (2007) aimed to develop a fault prediction model that ran on versions 2.0, 2.1 and 3.0 of Eclipse by constructing a fault database. Zhou and Leung (2006) studied the prediction of faults that had high and low priority using class-level metrics and logistic

regression, Naive Bayes, Random Forest, and nearest neighbor using generalization techniques on a KC1 dataset of NASA. They found that depth of inheritance tree and child amount are not important metrics and that the Chidamber–Kemerer metrics set is useful for software fault prediction. Menzies, DiStefano, Orrego, and Chapman (2004) studied fault predictors using method-level metrics and the Naive Bayes technique on the datasets of the PROMISE data repository. They used probability of detection, probability of false alarm and precision as evaluation criteria.

In last two years, studies on the software fault prediction have addressed the problem from two points: proposing new method or method combinations to increase prediction performance; using new parameters or feature selection approaches to propose most effective metrics for the prediction. Scanniello, Gravino, Adrian, and Menzies (2013) used multivariate linear regression to help selecting the independent variables which have statistically significant correlation with faultiness. The learning process was performed by clusters of related classes. In Rodríguez, Ruiz, Riquelme, and Harrison (2013), a descriptive approach based on subgroup discovery was proposed for software fault prediction. The induced rules for fault prediction were extracted from the datasets with the corresponding algorithms and then, the rules were applied to new instances for classification. Dejaeger, Verbraken, and Baesens (2013) showed that comprehensible networks with less nodes could be constructed using Bayesian Network (BN) classifiers to predict software faults. To achieve this goal, 15 different BN classifiers were used and results of the models were compared with the results of the models which employed other popular machine learning techniques. Furthermore, the effect of the Markov blanket principle on the prediction model performance was investigated and experiments show that there is no significant effect of the feature selection technique on the model performance. Oyetoyan, Cruzes, and Conradi (2013) used extended object-oriented metrics (cyclic dependencies) to identify defect profile of software components. To prove the effectiveness of the cyclic dependency metrics, the cyclic and non-cyclic metrics were used separately and analyses were done at the class levels and package levels in prediction models. Cotroneo, Natella, and Pietrantuono (2013) aimed to predict the location of Aging-Related Bugs (ARB) in complex software systems by using software complexity metrics as predictor variables and machine learning algorithms (NB, BN, DT, and LR with logarithmic transformation). After analyzing the bug reports of the projects which were used as dataset in the study, the complexity metrics were computed then, the metrics were utilized for predicting ARB-prone modules. Malhotra (2014) analyzed and compared different statistical (LR) and machine learning methods (LR, DT, ANN, Cascade Correlation Network, SVM, Group Method of Data Handling Method and Gene Expression Programming) for fault prediction. They were empirically validated to find the relationship between the static code metrics and the fault proneness of a module. In Couto, Pires, Valente, Bigonha, and Anquetil (2014), Granger causality test was used to evaluate whether past variations in source code metrics values can be used to forecast changes in time series of defects. In the first step of the study, threshold values for source code metrics were calculated. In the second step, the threshold values and the changed classes were taken as inputs to the defect prediction model and the model determined the faultiness state of the class. Czibula, Marian, and Czibula (2014) presented a novel classification model based on relational association rule mining for software defect prediction. To decrease the multi-dimensionality of the inputs, the relations between the features were analyzed by using Spearman's rank correlation coefficient in the data pre-processing phase then, a set of relations between the feature values and the rule set were defined. Khosgoftaar, Xiao, and Gao (2014) used rule-based (RB) models and case based learning (CBL) methods were gathered to classify the software modules as fault-prone or not fault-prone. The generated model (RB2CBL) used the genetic algorithm to optimize the model parameters. Laradji, Alshayeb, and Ghouti (2014) combined ensemble learning methods with feature selection models to identify software faultiness. Greedy forward selection was used as the feature selection method. The ensemble learning model used in the experiment included seven different classifiers and each classifier predicted the probability of faultiness. The output was the average of these probabilities.

Actually, the selections of dataset, evaluation criteria and predictive methods of previous studies were considered, and similar preferences were applied to give comparable results with the literature. In detail, there are 2 popular evaluation criteria to obtain experimental results such as ROC-AUC (e.g. Czibula et al. (2014), Dejaeger et al. (2013), Gondra (2008), Laradji et al. (2014), Mahaweerawat et al. (2006), Malhotra (2014)) and precision-recall (e.g. Cotroneo et al. (2013), Couto et al. (2014), Oyetoyan et al. (2013), Rodríguez et al. (2013)). Consequently, we employed the ROC-AUC for the performance evaluation to present comparable results with previous studies. At the same time, PROMISE repository is very popular for software fault prediction (e.g. Czibula et al. (2014), Dejaeger et al. (2013), Laradji et al. (2014), Malhotra (2014), Rodríguez et al. (2013)) and NASA projects with McCabe metrics in the PROMISE repository were used here. Additionally, method-level metrics and not object-oriented metrics were used because of the selected dataset properties. Selection of ANN and SVM is based on their common use for this problem (e.g. Gondra (2008), Kanmani, Uthariaraj, Sankaranarayanan, and Thambidurai (2004), Khoshgoftaar, Allen, Hudepohl, and Aud (1997), Laradji et al. (2014), Mahaweerawat et al. (2004), Malhotra (2014), Thwin and Quah (2003), Wang, Yu, and Zhu (2004), Xing et al. (2005)).

Following these studies this study presents the complementary results for the software fault prediction problem by employing ANFIS as a prediction method. Additionally, use of 3 parameters in McCabe metric set is proposed here as a result of ANFIS modeling process.

3. Metrics and data

The metrics used in software fault prediction can be separated into six groups: method-level metrics, class-level metrics, file-level metrics, component-level metrics, process-level metrics and different quantitative value-level metrics (Catal, 2011). McCabe metrics, which this study employs, correspond to the method level. Method-level metrics focus on programming concepts, and collecting them from the source codes developed using object-oriented or structural approaches is easy. Catal and Diri (2009) determined that predicted fault-prone modules are the methods likely to have faults during system testing or field tests when method-level metrics are used. As a result, method-level metrics are said to comprehensively address the programming effort. Moreover, Catal (2011) found that method-level metrics were commonly used between 1990 and 2007. All of these advantages enable this study to be comparable with the other studies presented in the literature. There are four types of McCabe metrics (http://promise.site.uottawa.ca/SERepository/datasets/cm1.arff):

- McCabe number of code line (loc)
- Cyclomatic complexity (v(g)): Amount of linear independent paths in a flow graph drawn for a module. Cyclomatic complexity changes according to decision nodes of the module structure. Testability decreases as long as cyclomatic complexity increases.
- Essential complexity (ev(g)): The value of the complexity of the graph that does not include D-prime graphs (graphs of sequence, selection and iteration operations) in the flow graph of a module. Maintainability and modularity decrease as long as essential complexity increases.

- Design complexity (iv(g)): Cyclomatic complexity of a module's reduced flow graph according to decisions and loops that do not have any calls to submodules. Maintainability and reusability decrease as long as design complexity increases.

In the study, the projects (CM1, JM1, KC1, KC2 and PC1) that have McCabe metrics in the PROMISE repository are selected and used to produce more reliable results. In this way, a dataset with 15,123 instances is generated and predictive models are implemented using 4 and 3 inputs (metrics) and one output (measured fault value).

To validate the predictive models, the *N*-fold cross validation is employed, which is the best known technique. The *N*-fold cross validation divides the data into *N* number of partitions, and each of them has an equal number of samples. Afterwards, training is performed with $(N-1)$ number of partitions, and the test is done with the remaining partitions. In this study, the prepared dataset is split into five parts ($N = 5$), indicating that 80% of the data is used for training and 20% is used for testing. Moreover, each partition of the dataset has the same faulty instance rate.

4. Method

Although some empirical methods were used to predict software faults, ANFIS, one of the soft computing techniques, has not been employed previously for such a purpose. In general, ANFIS is a supervised method that combines the advantages of a fuzzy inference system (FIS) and Artificial Neural Network (ANN) methods. In other words, ANFIS can construct an input–output mapping based on both human knowledge and stipulated input–output data pairs (Jang, 1993). The structure of fuzzy neural networks resembles the structure of normal neural networks with three or four hidden layers. Each layer in a fuzzy neural network is different and represents a task of a fuzzy inference system. The tasks are fuzzification, rule execution, normalization, and defuzzification, in order. The expert knowledge supplied to ANFIS models presented by Jang (1993) is numbers of fuzzy sets and types of membership functions. The rules are automatically built within the first-order Sugeno Type (if x is A and y is B than z = mx + ny + k), and rule weights, parameters of membership functions and ranges of fuzzy sets are adjusted through training and learning. A typical ANFIS architecture that consists of a total of six layers is presented in Fig. 1, and computation steps with implementation details are given as follows:

- Layer 0: enables taking crisp inputs from the environment; in this study, 3 and 4 input parameters based on McCabe metrics are employed.
- Layer 1: fuzzifies the inputs using specified fuzzy set numbers and membership functions. A primarily bell type membership function is employed but is optional. In this study, a bell-shaped function is used with 3 fuzzy sets for each input in experiments. Bell type membership has 3 parameters: center, width, and slope. They can be assumed to be the nonlinear antecedent parameters that are learned from the data.
- Layer 2: includes a specific node for each rule, and each node calculates the firing strength of the associated rule using the product operator. In this study, a total of 81 and 27 rules are generated and fired for 4 and 3 inputs respectively. Train data sets used in experiments consist of approximately 12,123 cases.
- Layer 3: normalizes the firing strengths of the each rule coming from layer 2.
- Layer 4: fires the rules using normalized strength values coming from layer 3 and calculates the contribution of each rule toward the overall output.
- Layer 5: contains only one node to produce crisp results by summing all of the results coming from layer 4.

An ANFIS model uses a hybrid learning algorithm that combines the least squares estimator and the gradient descent method (Jang, 1993). In each epoch, a least squares estimator is employed in the forward pass and the gradient descent method is employed in the backward pass. In the backward pass, ranges and parameters of membership functions are adjusted and, in the forward pass, coefficients in the polynomial expression are tuned (Sen, Sezer, Gokceoglu, & Yagiz, 2012). Training of the model continues until the minimum change in errors between the current and previous iteration is obtained or the desired epoch numbered is reached. In this study, a total of 100 iterations are used as stopping criteria.

ANN is derived from inspirations of the learning capability of the human brain. It offers an improved, generic and practical method to learn real, discrete or vector valued functions. ANN is tolerant of imperfect data in a training dataset and consists of a large number of units such as neurons and directed-weighted relations between these units (Lee & Park, 1992). As long as the number of units in a network grows, the complexity of the structure increases. In other words, the number of units specifies the complexity of the system. With these properties, ANN solves problems that cannot be solved using classical methods like the human brain does (Lee & Park, 1992).

The ANN models built in this study consists of 4 and 3 inputs (McCabe metrics), and one output (faultiness).The number of neurons in the hidden layer should be selected carefully according to suggested heuristics (Hecht-Nielsen, 1987; Kanellopoulas & Wilkinson, 1997; Yagiz, Sezer, & Gokceoglu, 2012). The ANN architecture suggested by Hecht-Nielsen (1987) is that $[\leqslant 2N_i + 1]$ where N_i refers to the number of inputs. In this study total 8 and 6 neurons are used in the hidden layers of the models which are built for 4 and 3 inputs. In other words, the number of neurons used herein satisfies the heuristic proposed by Hecht-Nielsen (1987). The training algorithm of the model is the scaled conjugate gradient. The scaled conjugate gradient method is a network training function that updates weight and bias values. The ANN model runs as a pattern recognition network. In other words, it is a feed forward network that can be trained to classify inputs according to target classes. The target data for our pattern recognition network consist of faultiness information on each software component. The details of the ANN are found in Negnevitsky (2005). The model performed the classification task as a result of approximately 50 iterations with 0.134 mean squared errors. The training process stops when the performance gradient falls below the minimum performance gradient. The ANN model built for 4 inputs in this study is shown in Fig. 2.

SVM is a classification method that can be used for both linear and nonlinear data. It uses nonlinear mapping to transform the training data into a higher dimension. It searches for the best linear separator hyperplane with the new dimension. SVM finds this hyperplane using support vectors and margins (Vapnik, 1998). Data from two classes can be separated by a hyperplane with a suitable nonlinear mapping to a high dimension (Fig. 3). The sequential minimal optimization (SMO) method is used to find the separating hyperplane for the SVM model. Memory consumption is controlled by the value that specifies the size of the kernel matrix cache (5000). The SMO algorithm stores only a submatrix of the kernel matrix, limited by the size specified by the size of the kernel matrix cache. The SVM model uses an optimization method to identify support vectors s_i, weights α_i, and the bias b, which are used to classify vectors x according to the following equation (http://www.mathworks.com/help/stats/svmtrain.html):

$$c = \sum \alpha_i k(s_i, x) + b$$

where k is a kernel function and a dot product. If $c \geqslant 0$, then x is classified as a member of the first group; otherwise, it is classified as a member of the second group.

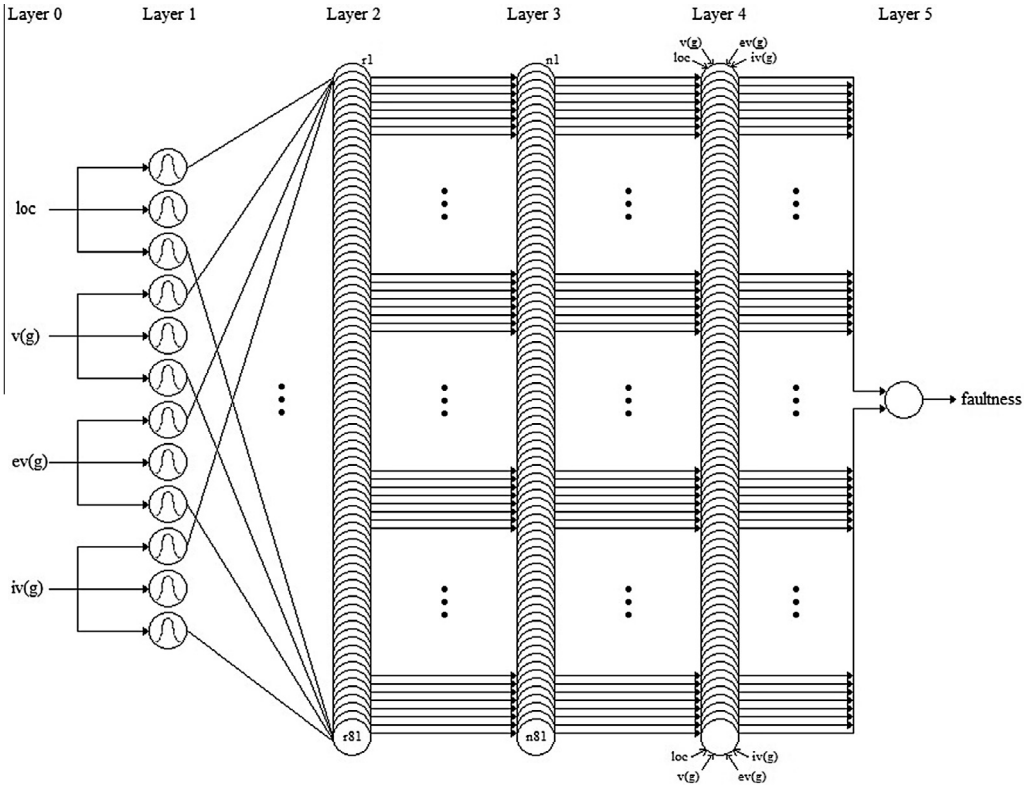

Fig. 1. Structure of ANFIS model.

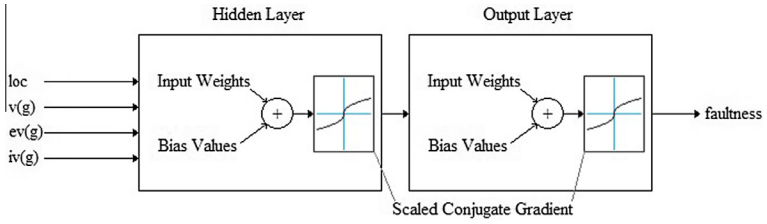

Fig. 2. ANN model structure.

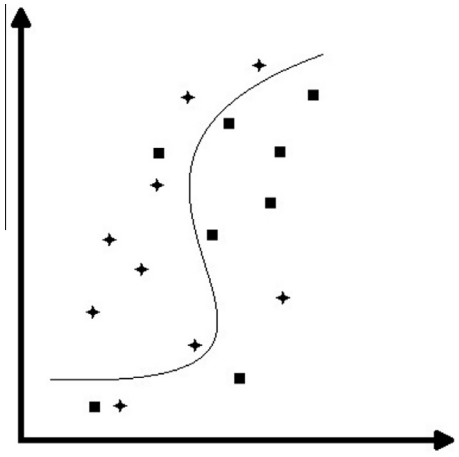

Fig. 3. Two classes separated by a hyperplane.

The training process may be slow; however, accuracy is usually high due to the capability of modeling nonlinear decision boundaries. SVM is suitable for both classification and prediction. The maximum iteration number is specified as 22,000, and no convergence is achieved within a lower number of iterations. More details on SVM can be found in (http://www.mathworks.com/help/stats/svmtrain.html).

The receiver operating characteristics (ROC) curves with the Area Under Curve (AUC) approach are widely used criteria for evaluating the performance of the prediction models. They are threshold-free criteria and show changes in true positive prediction rates versus false positive prediction rates. In other words, a ROC curve plots the true positive rates (TPR) on the *y*-axis and false positive rates (FPR) on the *x*-axis. As a result, a ROC curve starts at the points (0, 0) and reaches (1, 1). The ideal ROC curve passes through the point of (0, 1) with AUC = 1, indicating that there is no prediction error. An acceptable AUC value for a ROC curve should be higher than 0.5. A typical ROC curve is shown in Fig. 4 (Menzies, Greenwald, & Frank, 2007; Menzies et al., 2004). According to Catal (2011, 2012), the area under a ROC curve represents widely used criteria in disjoint datasets to evaluate the performance of machine learning algorithms in the software fault prediction problem.

5. Experiment

The objective of the experiments is to predict software faults using ANFIS, ANN and SVM and to compare the results of

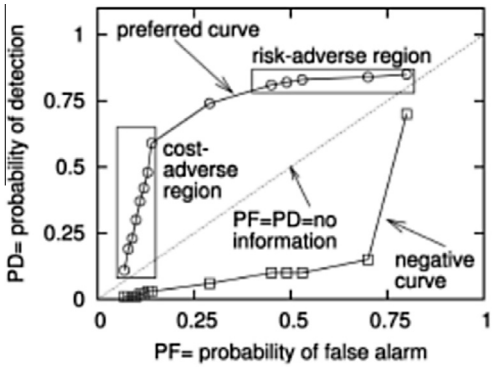

Fig. 4. Typical ROC curve [20, 21].

Table 1
ROC-AUC values for all datasets and methods with 4 McCabe metrics.

		AUC for SVM	AUC for ANN	AUC for ANFIS
1st Group	Train	0.6726	0.7513	0.7551
	Test	0.5939	0.6543	0.6525
2nd Group	Train	0.6644	0.7621	0.7509
	Test	0.596	0.6487	0.6433
3rd Group	Train	0.6582	0.7318	0.739
	Test	0.6223	0.6812	0.6856
4th Group	Train	0.6165	0.6921	0.6947
	Test	0.7788	0.8727	0.8457
5th Group	Train	0.6392	0.7279	0.7216
	Test	0.6434	0.7603	0.7564

the models. To achieve these objectives, three different models were built using the MATLAB 7.13.0 (R2011b) (MATLAB, 2009) environment. Then, the software fault prediction process was performed on the prepared training and test dataset derived from the PROMISE repository. Finally, ROC-AUC values of training and test sets of each model are calculated and compared.

In experiments, a total of 5 training and test datasets are produced. Train data sets consist of 12,123 cases and 2115 of them are faulty. It means that 17.45% of train data set is faulty cases. In test data sets, there are total 3000 cases and 550 of them are faulty. In other words, each test data set has 18.33% faulty cases. As a result, the amount and percentage of faulty cases are similar between different sets (test or train).

In first experiments, all McCabe metrics (loc, v(g), ev(g), and iv(g)) are used to be compatible with the previous experiments presented in literature (Table 1). Additionally, "ev(g)" parameter is extracted from the McCabe metric set, and second experiments are organized with the remaining parameters (loc, v(g), and iv(g)) (Table 2). The reason of this extraction is based on the observations on fault cases. When faulty cases are considered together, effect of "ev(g)" parameters is not sensible. In other words, there is almost no faulty cases which can be separated from the fault-free cases because of only its "ev(g)" value. The parameter of "loc" can be given as an opposite example to the parameter of "ev(g)". There are many faulty cases having different "loc" values from the fault-free cases even if they have similar "v(g), ev(g), and iv(g)" values. This observation requires the discussion about the use of "ev(g)" in modeling, because it may be inessential while the other McCabe metrics are used. As known, the use of inessential input parameters may affect the prediction performance in negative way and cause the increase in the complexity of models. Table 1 lists all achieved performance results in the first experiments which use 4 input parameters in ANN, ANFIS, and SVM models. Additionally, performance results obtained from second experiments are given in Table 2.

Table 2
ROC-AUC values for all datasets and methods with 3 McCabe metrics.

		AUC for SVM	AUC for ANN	AUC for ANFIS
1st Group	Train	0.6730	0.7533	0.7541
	Test	0.5931	0.6475	0.6515
2nd Group	Train	0.6644	0.7402	0.7508
	Test	0.5964	0.6457	0.6437
3rd Group	Train	0.6586	0.7262	0.7382
	Test	0.6219	0.6741	0.6904
4th Group	Train	0.6160	0.6944	0.6944
	Test	0.7795	0.8685	0.8573
5th Group	Train	0.6396	0.7450	0.7212
	Test	0.6449	0.7350	0.7579

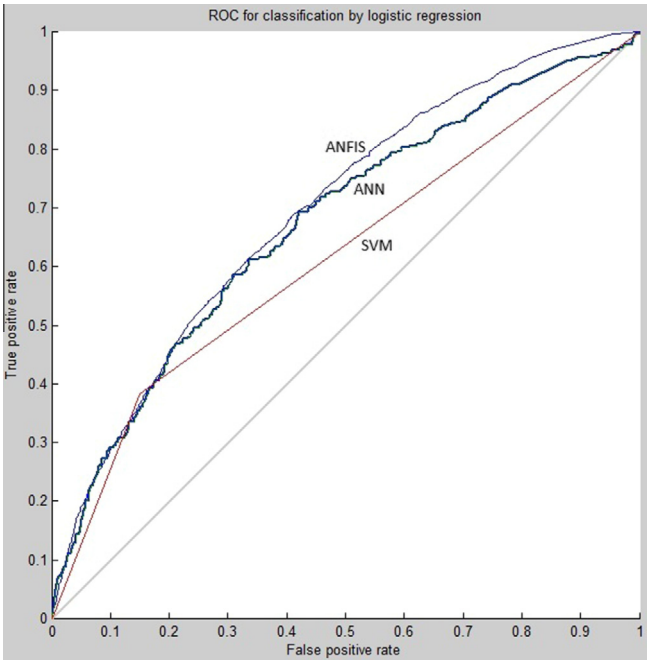

Fig. 5. ROC curves obtained from train set-4 (4 inputs).

The built models in both experiments are said to not overfit the training data sets because AUC values of the training sets are not much larger than AUC values of the test sets (Tables 1 and 2). In other words, the models have the generalization capability to compare them with one another.

All AUC values (Tables 1 and 2) are acceptable in this research area, and the results obtained from the SVM and ANN methods are in accordance with the results presented in the literature. In fact, they are used to clarify the appropriateness of the ANFIS method for the software fault prediction problem. In other words, the performance of the ANFIS should be compared with the other experimental methods. The model with SVM has the worst performance, whereas ANN and ANFIS more successfully predict faultiness (Tables 1 and 2). In fact, the best results of the first experiments are obtained from set-4 (SVM: 0.7788, ANN: 0.8724, ANFIS: 0.8457) and set 4 is most successful set for second experiments too (SVM: 0.7795 ANN: 0.8685 ANFIS: 0.8573). This means, set-4 has the greatest representing ability, and the most suitable set produces the predictive models that have the greatest generalizability. When ANN and ANFIS results are compared, the closeness between all of the results is observed (Tables 1 and 2).

The obtained ROC curves in the first experiments with 4 inputs are shown in Figs. 5 and 6 for a visual comparison. These results indicate that ANN and ANFIS are highly successful and that their

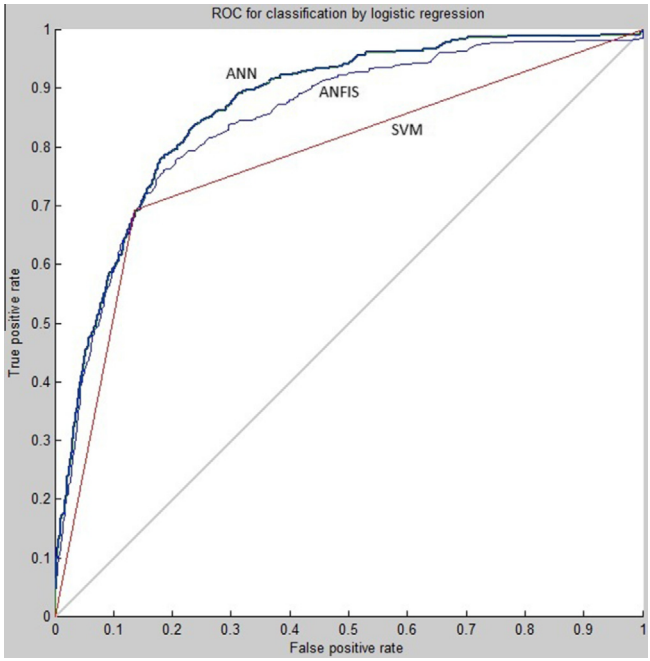

Fig. 6. ROC curves obtained from test set-4 (4 inputs).

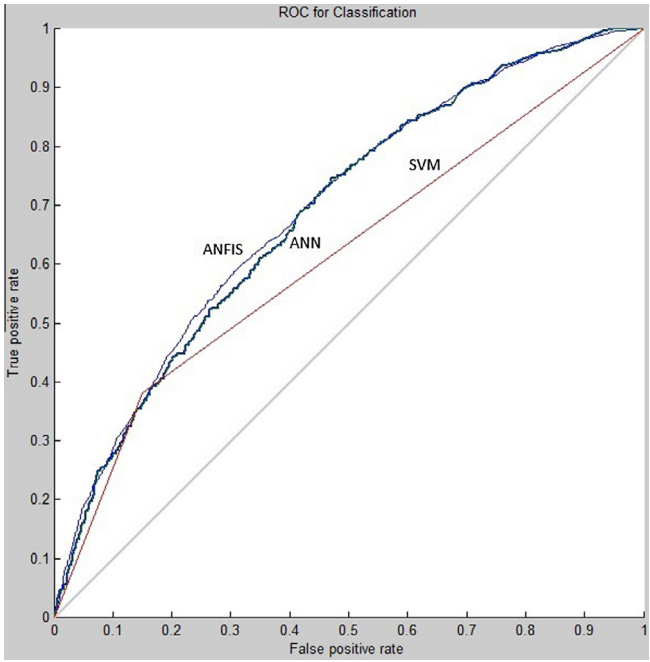

Fig. 7. ROC curves obtained from train set-4 (3 inputs).

performances are close to one another. According to these results, the best performance belongs to the model built using ANN. However, ANFIS must be considered to closely follow the ANN results. The model using SVM displays the worst performance, but all AUC values are acceptable in this research area.

The ROC curves obtained in the second experiments with 3 inputs are shown in Figs. 7 and 8. The performance values of the ANN and ANFIS methods are closer in these experiments then first ones. In other words, complexity of ANFIS models are decreased by extraction of "ev(g)" parameter and this change provides increase in performance of ANFIS models and also SVM's.

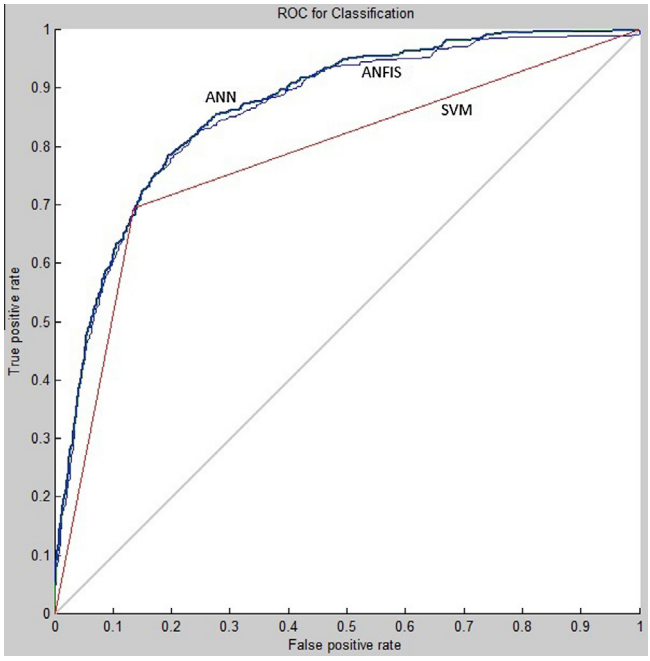

Fig. 8. ROC curves obtained from train set-4 (3 inputs).

6. Conclusion

Software fault prediction is seen as a highly important ability when planning a software project and much greater effort is needed to solve this complex problem using a nonlinear approach. Machine learning methods, which have been employed to predict software faultiness, have the ability of learning hidden relationships embedded in the data. In fact, they are dependent on the characteristics of the data and, when the data change, model can lose its performance dramatically. In software fault prediction, change of data means the change in project size, domain or software architecture and it is indispensable. To build the model which is less sensitive to change in data, use of expert knowledge with the data in learning stage becomes plausible suggestion. At this point, the hybrid approach of ANFIS becomes remarkable. The main motivation of this study is threefold: (1) propose ANFIS as one of the solution method for software fault prediction problem (2) to compare and assess the predictive performance of the ANFIS with the ANN and SVM which are the mostly employed methods on this problem (3) to make discussion on the McCabe parameter set and reduce the number of parameters which can be used while modeling.

The achieved performance by ANFIS with 3 parameters is 0.8573 and this result can convince us that ANFIS is a suitable method for software fault prediction. Unlike machine learning methods, ANFIS enables expert to handle vagueness of data more directly and it can be said that, domain expert of software engineering can be useful as much as data in learning process. To generalize this statement, more experiments on different data sets should be carried out and common architectural properties of ANFIS models should be extracted.

In software fault prediction, software metrics are proposed as a set (e.g. McCabe, Healstead, OOMetrics) and whole set is used in modeling with some predictive methods. In this study, we used the McCabe metrics and we proposed the use of 3 parameters (loc, v(g), and iv(g)) as a result of the modeling process of ANFIS. The parameters of the other metric sets should be considered but the most challenging point is that metrics for software fault prediction are proposed with the perspective of whole source codes of

the project will be developed. Namely, use of libraries, existing packages and framework abilities in software development process are ignored. However, misuse of these external codes may cause failures and faults can source from improper dependencies between inner and external software modules. This point will be focused to make closer software metrics to the development environments.

To make software fault prediction systems more practical is important, because cost of correcting faults in the later phases of the software development life cycle increases more and more. As a result, there is a need for a classifier which can be used in the early stages of the project development time to classify the module whether is faulty or not. Actually, the researchers and software practitioners can use the model proposed here in the early phases of the development. The earliest time of the software development cycle which the model can be used as classifier depends on the generalization ability of the model. In other words, the more generalizable models can be useful in the earlier phases of the development and ANFIS has the more advantageous to be general because of the expert knowledge supplied to it. However, similar researches should be done with large sized and different type of software projects to obtain generalized results.

References

Alsmadi, I., & Najadat, H. (2011). Evaluating the change of software fault behavior with dataset attributes based on categorical correlation. *Advances in Engineering Software, 42*(8), 535–546.

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

Catal, C. (2012). Performance evaluation metrics for software fault prediction studies. *Acta Polytechnica Hungarica, 9*(4), 193–206.

Catal, C., & Diri, B. (2009). Investigating effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences, 179*(8), 1040–1058.

Catal, C., Sevim, U., & Diri, B. (2011). Practical development of an eclipse-based software fault prediction tool using Naive Bayes algorithm. *Expert Systems with Applications, 38*(3), 2347–2353.

Cotroneo, D., Natella, R., & Pietrantuono, R. (2013). Predicting aging-related bugs using software complexity metrics. *Performance Evaluation, 70*(3), 163–178.

Couto, C., Pires, P., Valente, M. T., Bigonha, R. S., & Anquetil, N. (2014). Predicting software defects with causality tests. *Journal of Systems and Software, 93*, 24–41.

Czibula, G., Marian, Z., & Czibula, I. G. (2014). Software defect prediction using relational association rule mining. *Information Sciences, 264*, 260–278.

Daoming, G., & Jie, C. (2006). ANFIS for high-pressure waterjet cleaning prediction. *Surface & Coatings Technology, 201*(3), 1629–1634.

Dejaeger, K., Verbraken, T., & Baesens, B. (2013). Toward comprehensible software fault prediction models using bayesian network classifiers. *IEEE Transactions on Software Engineering, 39*(2), 237–257.

Gondra, I. (2008). Applying machine learning to software fault-proneness prediction. *Journal of Systems and Software, 81*(5), 186–195.

Hecht-Nielsen, R. (1987). Kolmogorov's mapping neural network existence theorem. In *Proceedings of the first IEEE international conference on neural networks* (pp. 11–14). San Diego, CA, USA.

Hu, Q. P., Xie, M., Ng, S. H., & Levitin, G. (2007). Robust recurrent neural network modeling for software fault detection and correction prediction. *Reliability Engineering & System Safety, 92*(3), 332–340.

Jang, J. S. R. (1993). ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man & Cybernetics, 23*(3), 665–685.

Kanellopoulas, I., & Wilkinson, G. G. (1997). Strategies and best practice for neural network image classification. *International Journal of Remote Sensing, 18*(4), 711–725.

Kanmani, S., Uthariaraj, V. R., Sankaranarayanan, V., & Thambidurai, P. (2004). Object oriented software quality prediction using general regression neural networks. *SIGSOFT Software Engineering Notes, 29*(5), 1–6.

Khoshgoftaar, T. M., Allen, E. B., Hudepohl, J. P., & Aud, S. J. (1997). Application of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks, 8*(4), 902–909.

Khosgoftaar, T. M., Xiao, Y., & Gao, K. (2014). Software quality assessment using a multi-strategy classifier. *Information Sciences, 259*, 555–570.

Laradji, I. H., Alshayeb, M., & Ghouti, L. (2014). Software defect prediction using ensemble learning on selected features. *Information and Software Technology*. http://dx.doi.org/10.1016/j.infsof.2014.07.005.

Lee, K. Y., & Park, J. H. (1992). Short-term load forecasting using an artificial neural network. *IEEE Transactions on Power Systems, 7*(1), 124–132.

Lo, S. P. (2003). An adaptive-network based fuzzy inference system for prediction of workpiece surface roughness in end milling. *Journal of Materials Processing Technology, 142*(3), 665–675.

Mahaweerawat, A., Sophatsathit, P., Lursinsap, C. & Musílek, P. (2004). Fault prediction in object-oriented software using neural network techniques. In *Proceedings of the InTech conference* (pp. 27–34). Houston, TX, USA.

Mahaweerawat, A., Sophatsathit, P., Lursinsap, C., & Musílek, P. (2006). MASP – An enhanced model of fault type identification in object-oriented software engineering. *Journal of Advanced Computational Intelligence and Intelligent Informatics, 10*(3), 312–322.

Malhotra, R. (2014). Comparative analysis of statistical and machine learning methods for predicting faulty modules. *Applied Soft Computing, 21*, 286–297.

MATLAB (2009). User's Guide Version 7.8, R2009a. MathWorks Co., USA <http://www.mathworks.com/help/stats/svmtrain.html>, MathWorks Co., USA, visit date 12.03.2013.

Menzies, T., DiStefano, J., Orrego, A. & Chapman, R. (2004). Assessing predictors of software defects. In *Proceedings of the Predictive software models workshop.* Chicago.

Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 32*(1), 2–13.

Najah, A. A., El-Shafie, A., Karim, O. A., & Jaafar, O. (2012). Water quality prediction model utilizing integrated wavelet-ANFIS model with cross-validation. *Neural Computing & Applications, 21*(5), 833–841.

Negnevitsky, M. (2005). *Artificial intelligence a guide to intelligent systems* (2nd ed.). Harlow, England: Addison-Wesley.

Oyetoyan, T. D., Cruzes, D. S., & Conradi, R. (2013). A study of cyclic dependencies on defect profile of software components. *Journal of Systems and Software, 86*(12), 3162–3182.

Perez, J. A., Gonzalez, M., & Dopico, D. (2010). Adaptive neurofuzzy ANFIS modeling of laser surface treatments. *Neural Computing & Applications, 19*(1), 85–90.

Pradhan, B., Sezer, E. A., Gokceoglu, C., & Buchroithner, M. F. (2010). Landslide susceptibility mapping by neuro-fuzzy approach in a landslide prone area (Cameron Highland, Malaysia). *IEEE Transactions on Geoscience and Remote Sensing, 48*(12), 4164–4177.

Promise Software Engineering Repository Public Datasets (2013). <http://promise.site.uottawa.ca/SERepository/datasets/cm1.arff>.

Rodríguez, D., Ruiz, R., Riquelme, J. C., & Harrison, R. (2013). A study of subgroup discovery approaches for defect prediction. *Information and Software Technology, 55*(10), 1810–1822.

Sayyad, S. & Menzies, T. (2005). The PROMISE Repository of Software Engineering Databases. Canada: University of Ottawa, <http://promise.site.uottawa.ca/SERepository>.

Scanniello, G., Gravino, C., Adrian, M. & Menzies, T. (2013). Class level fault prediction using software clustering. In *Proceedings of the 28th international conference on automated software engineering* (pp. 640–645), Silicon Valley, CA.

Sen, S., Sezer, E. A., Gokceoglu, C., & Yagiz, S. (2012). On sampling strategies for small and continuous data with the modeling of genetic programming and adaptive neuro-fuzzy inference system. *Journal of Intelligent & Fuzzy System, 23*(6), 297–304.

Sezer, E. A., Pradhan, B., & Gokceoglu, C. (2011). Manifestation of an adaptive neuro-fuzzy model on landslide susceptibility mapping: Klang valley, Malaysia. *Expert Systems with Applications, 38*(7), 8208–8219.

Thomas, J. (1976). McCabe, a complexity measure. *IEEE Transactions on Software Engineering, 2*(4), 308–320.

Thwin, M. M.m & Quah, T. (2003). Application of neural networks for software quality prediction using object-oriented metrics. In *Proceedings of the 19th international conference on software maintenance (113–122).* Amsterdam, The Netherlands.

Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., & Haesen, R. (2008). Mining software repositories for comprehensible software fault prediction models. *Journal of Systems and Software, 81*(5), 823–839.

Vapnik, V. (1998). The support vector method of function estimation. In J. A. K. Suykens & J. Vandewalle (Eds.), *Nonlinear modeling: Advanced black-box techniques* (pp. 55–85). Boston: Kluwer Academic Publishers.

Wang, Q., Yu, B., & Zhu, J. (2004). Extract rules from software quality prediction model based on neural network. In *Proceedings of the 16th IEEE international conference on tools with artificial intelligence* (pp. 191–195). Boca Raton, FL, USA: IEEE Computer Society.

Xing, F., Guo, P., & Lyu, M. R. (2005). A novel method for early software quality prediction based on support vector machine. In *Proceedings of the 16th IEEE international conference on software reliability engineering* (pp. 213–222). Chicago, IL.

Yagiz, S., Sezer, E. A., & Gokceoglu, C. (2012). Artificial neural networks and nonlinear regression techniques to assess the influence of slake durability cycles on the prediction of uniaxial compressive strength and modulus of elasticity for carbonate rocks. *International Journal for Numerical and Analytical Methods in Geomechanics, 36*(14), 1636–1650.

Zhou, Y., & Leung, H. (2006). Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Transactions on Software Engineering, 32*(10), 771–789.

Zimmermann, T., Premraj, R., & Zeller, A. (2007). Predicting defects for Eclipse. In *Proceedings of the 3rd international workshop on predictor models in software engineering, May 20–26* (p. 9). Minneapolis, MN, USA.