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.

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/

- 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