Applied Soft Computing 21 (2014) 286–297

Contents lists available at ScienceDirect

Applied Soft Computing

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

Comparative analysis of statistical and machine learning methods for predicting faulty modules

Ruchika Malhotra

Department of Software Engineering, Delhi Technological University, Bawana Road, Delhi 110042, India

ARTICLE INFO

Article history:
Received 25 October 2011
Received in revised form 26 January 2014
Accepted 22 March 2014
Available online 31 March 2014

Keywords:
Software quality
Static code metrics
Logistic regression
Machine learning
Receiver Operating Characteristic (ROC) curve

ABSTRACT

The demand for development of good quality software has seen rapid growth in the last few years. This is leading to increase in the use of the machine learning methods for analyzing and assessing public domain data sets. These methods can be used in developing models for estimating software quality attributes such as fault proneness, maintenance effort, testing effort. Software fault prediction in the early phases of software development can help and guide software practitioners to focus the available testing resources on the weaker areas during the software development. This paper analyses and compares the statistical and six machine learning methods for fault prediction. These methods (Decision Tree, Artificial Neural Network, Cascade Correlation Network, Support Vector Machine, Group Method of Data Handling Method, and Gene Expression Programming) are empirically validated to find the relationship between the static code metrics and the fault proneness of a module. In order to assess and compare the models predicted using the regression and the machine learning methods we used two publicly available data sets AR1 and AR6. We compared the predictive capability of the models using the Area Under the Curve (measured from the Receiver Operating Characteristic (ROC) analysis). The study confirms the predictive capability of the machine learning methods for software fault prediction. The results show that the Area Under the Curve of model predicted using the Decision Tree method is 0.8 and 0.9 (for AR1 and AR6 data sets, respectively) and is a better model than the model predicted using the logistic regression and other machine learning methods.

1. Introduction

As the size and complexity of the software is increasing day by day, it is usual to produce software with faults. The identification of faults in a timely manner is essential as the cost of correcting these faults increases exponentially in the later phases of the software development life cycle. Hence, testing is a very expensive process and one-third to one-half of the overall cost is allocated to the software testing activities [47]. There are several approaches being proposed to detect the faults in the early phases of the software development [47]. This motivates the development of fault prediction models that can be used in classification of a module as fault prone or not fault prone.

Several static code metrics have been proposed in the past to capture various aspects in the design and source code, for example [17,22,35]. These metrics can be used in developing the fault prediction models. The prediction models can then be used by the software organizations during the early phases of software development to identify faulty modules. The software organizations can use these subset of metrics amongst the available large set of software metrics. The metrics are computed from the proposed model to obtain the information about the quality of the software and can be assessed in the early stages of software development. These quality models will allow the researchers and software practitioners to focus the available testing resources on the faulty areas of the software, which will help in producing a improved quality, low cost and maintainable software. The static code metrics have been advocated as widely used measures in the literature [9,36,37].

The machine learning (ML) methods have been successfully applied over the years on a range of problem domains such as medicine, engineering, physics, finance and geology. These methods have also started being used in solving the classification and control problems [1,11–13,34,56]. In the existing literature, the researchers have used various methods to establish the relationship between the static code metrics and fault prediction. These methods include the traditional statistical methods such as logistic regression (LR) [8,26,32,39] and the machine learning (ML) methods such as decision trees [16,25,27,29,30,33,42,46,49,52], Naïve

E-mail address: ruchikamalhotra2004@yahoo.com

http://dx.doi.org/10.1016/j.asoc.2014.03.032
1568-4946/

Bayes [6,9,10,38,41,49], Support Vector Machines [52,54], Artificial Neural Networks [28,33,38]. However, few studies compare the LR models with the ML models for software fault prediction using the static code metrics. Hall et al. [18] concluded that more quality studies should be conducted for software fault prediction using ML methods and Menzies et al. [36] advocated the use of public data sets for software fault prediction. It is natural for the software practitioners and potential users to wonder, "Which ML method is best?", or more realistically, "Is the predictive capability of the ML methods comparable or better than the traditional LR methods?". For this reason, this paper compares the model predicted using the LR method with the models predicted using the widely used ML methods and rarely used ML methods (Cascade Correlation Network (CCN), Group Method of Data Handling Polynomial Method (GMDH), Gene Expression Programming (GEP)). The evidence obtained from the data based empirical studies can help the software practitioners and researchers in obtaining the most powerful support for accepting/rejecting a given hypothesis [2]. Hence, conducting empirical studies to compare the models predicted using the LR and the ML methods is important to develop adequate body of knowledge so that the well formed and widely accepted theories can be produced.

The main motivation of the paper is threefold: (1) to show the performance of the ML methods such as Support Vector Machines (SVM), Decision Tree (DT), CCN, GMDH and GEP for software fault prediction; (2) to compare and assess the predictive performance of the models predicted using the ML methods with the model predicted using the LR methods; and (3) to evaluate the performance capability of the ML methods using public data sets, i.e. across two systems.

Thus, in this work we (1) build fault proneness models and (2) empirically compare the results of the LR and the ML methods. In this paper, we investigate the below issues:

1. How accurately and precisely do the static code metrics predict faulty modules?
2. Is the performance of the ML methods better than the LR method?

The validation of the models predicted using the LR and the ML methods is carried out using Receiver Operating Characteristic (ROC) analysis. We use the ROC curves to obtain the optimal cut off point that provides balance between the faulty and non faulty modules. The performance of the models constructed to predict faulty or non faulty modules is evaluated using the Area Under the Curve (AUC) obtained from the ROC analysis [11]. In order to perform the analysis we validate the performance of these methods using public domain AR1 and AR6 data sets. The AR1 and AR6 data sets consist of 121 and 101 modules, respectively. These data sets were developed using C language [35]. The data are obtained from the Promise data repository [35] and collected by Software Research Laboratory (Softlab), Bogazici University, Istanbul, Turkey [19]. Thus, the main contributions of the paper are: first, we performed the comparative analysis of the models using the LR method with the models predicted using the ML methods for prediction of faulty modules. Second, we analyze public domain and industrial data sets, hence analysing valuable data in an important area. Third, we analyze six ML methods and apply ROC analysis to determine their effectiveness.

The paper is organized as follows: Section 2 presents the research background that summarizes the static code metrics included and describes the sources from which the data is collected. Section 3 describes the descriptive statistics and the performance measures used for model evaluation. The results of model prediction are presented in Section 4 and the models are validated in Section 5. Section 6 summarized the threats to validity of the models and the conclusions of the work are given in Section 7.

2. Research background

In this section we present the dependent and independent variables used in this paper (Section 2.1). We also describe the data collection procedure in Section 2.2.

2.1. Dependent and independent variables

Fault proneness is the binary dependent variable in this work. Fault proneness is defined as the probability of fault detection in a module [2,4,7]. We use the LR method, which is based on probability prediction. The binary dependent variable is based on the faults that are found during the software development life cycle.

For this study, we predict fault prone modules from static code metrics defined by Halstead [17], and McCabe [35]. The software metrics selected in this paper are procedural and module based metrics, where a module is defined as the smallest individual unit of functionality. We find the relationship of the static code metrics with fault proneness since they are "useful", "easy to use", and "widely used" metrics [36].

First, the static code metrics are known as useful as they have shown higher probability of fault detection in past [36]. The results in this study also show that the correctly predicted percentage of faulty modules was high. Second, the metrics like lines of code and the metrics given by Halstead and McCabe [17,35] can be computed easily and at low cost, even for very large systems [30]. Third, many researchers have used the static code metrics in the literature [8,25–30,32,33,38–42,46,49,52–54]. It has been stated in [36] that "Verification and Validation textbooks advise using static code complexity metrics to decide which modules are worthy of manual inspections". Table 1 presents the static code metrics chosen in this study.

2.2. Empirical data collection

This study makes use of two public domain data sets AR1 and AR6 available in the Promise data repository [45] and donated by Software Research Laboratory (Softlab), Bogazici University, Istanbul, Turkey [23]. The data in AR1 and AR6 are collected from embedded software in a white-goods product. The data was collected and validated by the Prest Metrics Extraction and Analysis Tool [44] available at http://softlab.boun.edu.tr/?q=resources&i=tools. The data in AR1 and AR6 was implemented in the C programming language. Both the data sets were collected in 2008 and donated by Softlab in 2009. The AR1 system consists of 121 modules (9 faulty/112 non faulty). The AR6 system consists of 101 modules (15 faulty/86 non faulty). Both the data sets comprise of 29 static code attributes (McCabe, Halstead and LOC measures) and 1 fault information (false/true). Table 2 summarizes the distribution of faulty modules in the AR1 and AR6 data sets. The table shows that 7.44% of modules were faulty in the AR1 data set and 14.85% of modules were faulty in the AR6 data set.

3. Research methodology

In this section, steps taken to analyze the static code metrics are described.

3.1. Descriptive statistics and outlier analysis

Before further analysis can be carried out, the data set must be suitably reduced by analysing it and then drawing meaningful