Applied Soft Computing 13 (2013) 189–200

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

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

Hybrid intelligent systems for predicting software reliability

Ramakanta Mohanty[a], V. Ravi[b,*], M.R. Patra[c]

[a] Computer Science Department, Keshav Memorial Institute of Technology, Narayanaguda, Hyderabad 500029, India
[b] Institute for Development and Research in Banking Technology, Castle Hills Road #1, Masab Tank, Hyderabad 500057, AP, India
[c] Computer Science Department, Berhampur University, Berhampur 760007, Orissa, India

ARTICLE INFO

Article history:
Received 13 July 2010
Received in revised form 12 July 2012
Accepted 4 August 2012
Available online 17 August 2012

Keywords:
Software reliability
Multiple Linear Regression (MLR)
Multivariate Adaptive Regression Splines (MARS)
Back Propagation Neural Network (BPNN)
Counter Propagation Neural Network (CPNN)
Dynamic Evolving Neuro-Fuzzy Inference System (DENFIS)
TreeNet
Group Method of Data Handling (GMDH)
Genetic Programming (GP)
Recurrent architecture and ensemble model

ABSTRACT

In this paper, we propose novel recurrent architectures for Genetic Programming (GP) and Group Method of Data Handling (GMDH) to predict software reliability. The effectiveness of the models is compared with that of well-known machine learning techniques viz. Multiple Linear Regression (MLR), Multivariate Adaptive Regression Splines (MARS), Backpropagation Neural Network (BPNN), Counter Propagation Neural Network (CPNN), Dynamic Evolving Neuro-Fuzzy Inference System (DENFIS), TreeNet, GMDH and GP on three datasets taken from literature. Further, we extended our research by developing GP and GMDH based ensemble models to predict software reliability. In the ensemble models, we considered GP and GMDH as constituent models and chose GP, GMDH, BPNN and Average as arbitrators. The results obtained from our experiments indicate that the new recurrent architecture for GP and the ensemble based on GP outperformed all other techniques.

1. Introduction

During the past decade increasing attention was focused on computer software. As computing systems became more numerous, more complex, and more deeply embedded in modern society, there is a pressing need for developing systematic approaches to software development and software maintenance. Hence, software engineering, which deals with all these aspects, has become an independent area of research and practice. As reported in Ref. [13], "The software crisis is a term used to describe the recurring system development problems where software problems cause the system to be late, over cost, and/or not responsive to user's needs and requirements". Software engineering was focused on the tools, techniques, and methods necessary to solve the "software crisis." Papers and reports abound with description of new methodologies and software tools developed in university and corporation laboratories. Much of this discussion has centered on what is sometime termed a "silver bullet" i.e. the tool or technique that can solve the software crisis.

Gibbs [13] proposed that such a silver bullet exists and it is a software engineering project management issue. A properly managed project, in a mature software engineering environment, managed by a competent manager, can repeatedly deliver a software system on time, within cost, and satisfactory to the user. Good project management environment can compensate when things go wrong, for example, adjust the delivery schedule under changing conditions, select appropriate people for the job at hand, provide clear and unambiguous direction, monitor progress and job completion status, and take appropriate actions when controlling metrics indicate plans are not being followed. Thus software reliability plays an important role in software project management.

Software engineering is inherently knowledge intensive. Software processes and products are human centered [30]. In an attempt to pursue fruitful developments it is beneficial to identify sources of such shortcomings. At the same time they dealt with high-level abstract concepts and result in constructs whose functioning is not governed by the requirement of the laws of physics. There is no direct use of ideas such as continuity that is very intuitive and helpful in the physical world. In software engineering, small changes to the requirements may result in far drastic and

* Corresponding author. Tel.: +91 40 2353 4981; fax: +91 40 2353 5157.
E-mail addresses: ramakanta5a@gmail.com (R. Mohanty), rav_padma@yahoo.com (V. Ravi), mrpatra12@gmail.com (M.R. Patra).

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2012.08.015

radical changes in a total cost of the overall project. There is no concept of time because software products do not wear out in contrast to physical systems that are subjected to such deterioration.

Reliability is probably the most important of the characteristics inherent in the concept "software quality". Software reliability is defined as the probability that the software will work without failure for a specified period of time [37]. Software reliability is an important factor related to defects and faults. It differs from hardware reliability in that it reflects the design perfection, rather than manufacturing perfection. The principal factors that affect software reliability are (i) fault introduction, (ii) fault removal and (iii) the environment. Fault introduction depends primarily on the characteristics of the product and the development process. The characteristics of development process include software engineering technologies and tools used the level of experience of the personnel, volatility of requirements, and other factors. Failure discovery, in turn, depends on the extent to which the software has been executed and the operational profile. Because some of the foregoing factors are probabilistic in nature and operate over time, software reliability models have generally been formulated in terms of random processes in execution time.

In the past few years much research work has been carried out in software reliability and forecasting but no single model could capture software characteristics very accurately.

In this paper, we propose recurrent architecture for GP and GMDH and also ensemble models involving GP, GMDH as constituents and GP, GMDH, BPNN and Average as arbitrators in predicting software reliability. We compared their performance with that of some of the well-known machine learning techniques.

The rest of the paper is organized in the following manner. In Section 2, a brief review of the works carried out in area of software reliability prediction is presented. In Section 3, various stand-alone machine-learning techniques applied in this paper are briefly described. In Section 4, the experimental design followed in this paper is presented, while Section 5 presents our proposed methodology. It is followed by Section 6 that discusses the results and discussion. Finally, Section 7 concludes the paper.

2. Literature survey

Given the importance of software reliability in software engineering, its prediction becomes a very critical issue. Intelligent and soft computing techniques have been dominating in the last two decades. The recently published comprehensive state-of-the-art review [34] justifies this issue.

In recent years, neural networks (NN) have proven to be universal approximators by successfully modeling any non-linear continuous function with arbitrary degree of accuracy [5,30,52]. Many papers published in the literature confirmed that neural networks could offer promising approaches to software reliability estimation and modeling [5,6,8,20,23–27,46,48–51]. For example, Sherer [46] applied neural networks to predict software faults in several NASA projects. She found that if software faults tend to cluster, then identification of fault-prone modules through initial testing could guide subsequent testing efforts that focus on these modules. Cai et al. [5] presented a review on software reliability modeling. The review discussed different types of probabilistic software reliability models and their shortcomings. They developed a simple yet powerful fuzzy software reliability model, which was powerful alternative to other methods. Besides, Sitte [47] compared the predictive performance of two different methods of software reliability prediction: 'neural networks' and 'recalibration for parametric models'. She found that neural networks are not only much simpler to use than the recalibration method, but also are better trend predictors. It is noted that Karunanithi et al. [19–21] first applied neural network architecture to estimate the software reliability and used the execution time as input, the cumulative number of detected faults as desired output, and encoded the input and output into the binary bit string. Furthermore, they also illustrated the usefulness of NN models for software reliability growth prediction and showed that the NN approach is capable of developing models of varying complexity. Khoshgoftaar et al. [25] and Khoshgoftaar and Szabo [24] used the neural network as a tool for predicting the number of faults in programs. They introduced an approach for static reliability modeling and concluded that the neural networks produce models with better quality of fit and predictive quality. Most of the neural networks used for software reliability modeling can be classified into two classes. One class used cumulative execution time as inputs and the corresponding accumulated failures as desired outputs. This class focuses on modeling software reliability by varying different kinds of neural network including recurrent neural network [19,21,47,50,51]. The other class of neural networks model the software reliability based on multiple-delayed input–single-output neural network. Tian and Noore [50] proposed an evolutionary neural network modeling approach for software cumulative failure time prediction using multiple-delayed-input–single-output architecture. However, the problem with these approaches is that we have to predetermine the network architecture such as the number of neurons in each layer and also the number of the layers. Later, Rajkiran and Ravi [43] developed an ensemble model, which accurately forecast software reliability. They used MLR, MARS, Dynamic Evolving Neuro-Fuzzy Inference System (DENFIS) and TreeNet as constituents to develop the ensemble. They designed and tested three linear and one non-linear ensembles. They concluded that the non-linear ensemble outperformed all other ensembles and also the constituent's statistical and intelligent techniques. Further, Rajkiran and Ravi [44] proposed the use of Wavelets Neural Networks (WNN) with Morlet and Gaussian wavelet transfer functions to predict software reliability. Its performance was compared to that of Multiple Linear Regression (MLR), Multivariate Adaptive Regression Splines (MARS), MLP, Threshold accepting trained neural network (TANN), Pi-Sigma Network (PSN), GRNN, Dynamic Evolving Neuro-Fuzzy Inference System (DENFIS) and TreeNet in terms of normalized root mean square error (NRMSE) obtained on test data. They reported that the WNN outperformed all other techniques. Then, Ravi et al. [45] proposed the use of threshold accepting trained wavelet neural network (TAWNN) to predict operational risk in banks and firms by predicting software reliability. They observed that WNN based models outperformed TAWNN and other techniques over all the lags. Aljahdali and Telbany [1] employed Genetic Algorithm (GA) to solve the multi-objective optimization problems involving the objectives of NRMSE and correlation coefficient obtained by ensembling different auto regressive models to predict software reliability. Pai and Hong [39] applied Support Vector Machines (SVMs) for forecasting software reliability where Simulated Annealing (SA) algorithm was used to select the parameters of the SVM model.

Most recently, Mohanty et al. [32,33] employed Group Method of Data Handling (GMDH) and Genetic Programming (GP) to predict software reliability. They compared the GMDH and GP with that of MLR, MARS, MLP, TANN, PSN, GRNN, DENFIS and TreeNet in terms of NRMSE obtained on test data. They found that GP outperformed all other techniques.

3. Overview of the techniques applied

Here we present a brief overview of the machine learning, soft computing and statistical techniques that are employed in this