Extract Rules from Software Quality Prediction Model Based on Neural Network

Qi Wang

Dept. of Electronic Engineering
Shanghai Jiaotong University

wangqi_sjtu@sjtu.edu.cn

Bo Yu

Dept. of System Verification
Lucent Technologies Optical Networks Co., Ltd Shanghai
boyu@lucent.com

Jie Zhu

Dept. of Electronic Engineering
Shanghai Jiaotong University

zhujie@sjtu.edu.cn

Abstract

To get a highly reliable software product to the market on schedule, software engineers must allocate resources on the fault-prone software modules across the development effort. Software quality models based upon data mining from past projects can identify fault-prone modules in current similar development efforts. So that resources can be focused on fault-prone modules to improve quality prior to release. Many researchers have applied the neural networks approach to predict software quality. Although neural networks have shown their strengths in solving complex problems, their shortcoming of being 'black boxes' models has prevented them from being accepted as a common practice for fault-prone software modules prediction. That is a significant weakness, for without the ability to produce comprehensible decisions, it is hard to trust the reliability of neural networks that address real-world problems. In this paper, we introduce an interpretable neural network model for software quality prediction. First, a three-layer feed-forward neural network with the sigmoid function in hidden units and the identity function in output unit was trained. The data used to train the neural network is collected from an earlier release of a telecommunications software system. Then use clustering genetic algorithm (CGA) to extract comprehensible rules from the trained neural network. We use the rule set extracted from the trained neural network to detect the fault-prone software modules of the later release and compare the predicting results with the neural network predicting results. The comparison shows that although the rule set's predicting accuracy is a little less than the trained neural network, it is more comprehensible.

1. Introduction

To be profitable, a software development effort must get a high quality product to the market in a reasonable time. To get a high quality product to the market on schedule, developers must know which modules are likely to have faults, so that resources can be focused on *fault-prone* modules to improve quality prior to releases. The modeling problem at hand is to classify modules as *fault-prone*, or not. Various classification techniques have been applied to this problem, including logistic regression [1], optimal set reduction [2], fuzzy classification [3], classification trees [4]. Compared with these techniques, artificial neural networks are adept at modeling nonlinear functional relationship, and thus, are attractive for software quality modeling. Many researchers have applied the neural networks approach to predict software quality [5,6,7,8]. Most of these investigations have focused on the accuracy of the approach when compared to other classification techniques. But the model that this paper introduces takes the network's comprehensibility besides the predicting accuracy into consideration.

Many researchers are hesitant to use neural nets because of their shortcoming of being "black boxes" . That is a significant weakness, for without the ability to produce comprehensible decisions, it is hard to trust the reliability of neural networks that address real-world problems. Thus, the explanation and interpretation of the knowledge stored in the architecture and the weights of the neural nets are very important to gain the acceptance of practitioners. Our proposed idea comprises mainly the use of a genetic clustering method to extract IF-THEN rules from the trained neural network. The IF part of the rule specifies a conjunction of conditions on predicting attribute values and the THEN part specifies a predicted value for the goal attribute.

This paper is organized as follows. Section 2 is devoted to describing the clustering genetic algorithm. In section 3, the modeling methodology is discussed. In section 4, a case study of a telecommunications system developed by Lucent Technology Optical Network Co. is presented, illustrating how this approach can be applied. Then, this paper is finished with some conclusions and our future work.

2. Clustering genetic algorithm for rule extraction from neural networks

The clustering genetic algorithm is a kind of decompositional algorithm. It is desirable to find an optimal solution to the clustering of the hidden units activation values in order to get acceptable rule sets. The genetic algorithms are widely believed to be effective on such kind of problems [12]. This approach basically consists of two steps:

1) Apply a clustering algorithm to find clusters of hidden units activation values for each class;
2) Generate a set of rules to describe the hidden units discretized values in relation to the inputs.

The rule sets generated are in the following form:

IF $\{v_{\min}^{1} \le h_1 \le v_{\max}^{1}$ and… and $v_{\min}^{m} \le h_m \le v_{\max}^{m}\}$ THEN class C_j;

Where h_is are the hidden units activation expressions, as a function of the input units and v^j are the maximum or minimum values from the obtained clusters for each class C_j. Our clustering genetic algorithm is different with the approach described in [12]. In the following part, we'll present our approach detailedly.

2.1. Individual representation

Considering that there are N objects to be clustered, denoted as $\{O_1,O_2,\ldots,O_N\}$, a genotype is represented as a string of length N. Each string represents a subset of $\{O_1,O_2,\ldots,O_N\}$. If O_i is in the subset, the i^{th} position of the string will be 1; Otherwise, the i^{th} bit will be 0. The n strings are initia led in such a way that the numbers of 1's in the strings almost uniformly distributed within $[1,N]$. Each string represents the seeds for a clustering because we will use each object in the subset represented by this string as a seed to generate a cluster. Let $T=\{T_1,T_2,\ldots,T_m\}$ be such a subset. Let initial clusters $C_i=\{T_i\}$ for $i=1,2,\ldots,m$ and initial centers of clusters $S_i=T_i$ for $i=1,2,\ldots,m$. The generation of the clusters proceeds as follows:

1) The objects in $\{O_1,O_2,\ldots,O_N\}$-T are taken one by one and the distance between the taken object O_i and each S_j is calculated using the following distance function.

$$d(O_i,S_j)=\sum_{q=1}^{p}\left|O_{jq}-S_{jq}\right| \qquad (1)$$

Where $O_i=(o_{i1},o_{i2},\ldots,o_{ip})$ and $S_j=(s_{j1},s_{j2},\ldots,s_{jp})$, p is the number of hidden units.

2) Then, O_i belongs to C_j if $d(O_i,S_j) = d(O_i,S_k)$ for all k such that $1= k = m$ and k ? j.
3) If O_i is classified as in cluster C_j, the center S_j of the cluster C_j will be recomputed using the following function when O_i is added to C_j.

$$S_{jq}=\frac{\sum_i o_{iq}}{\left|C_j\right|} \qquad (2)$$

Where the summation is over all i such that $o_i \in C_j$ and $\left|C_j\right|$ is the number of objects in C_j.

After all objects in $\{O_1,O_2,\ldots,O_N\}$-T have been considered, we will obtain m clusters $C_1,C_2,\ldots,C_m$ with centers $S_1,S_2,\ldots,S_m$. Each cluster C_j is generated by the seed T_j. We define $\{C_1,C_2,\ldots,C_m\}$ to be the set of clusters generated by this string.

2.2. Fitness function

The fitness function must reflect the way each individual is, in comparison to the others, becoming closer to the optimal solution. The calculation of the fitness is a very important part of our algorithm. The fitness of string R is the sum of two scores, namely, *SCORE1* and *SCORE2*. Let $\{C_1,C_2,\ldots,C_m\}$ be the set of clusters generated by string R. Let C_j' be defined as follows.

$$C_j' = \{\, O_i \mid O_i \in C_j \text{ and } d(O_i,S_j) = d(O_i,S_k) \text{ for all } k \text{ such that } 1= k = m \text{ and } k \neq j\}$$

Note that C_j' is a subset of C_j and contains those subjects of C_j that are indeed closer to the center of C_j than to other centers. Also note that C_j' may be a proper subset of C_j. In other words, there may be some objects of C_j that are closer to other centers than to the center of C_j. Now, *SCORE1* is defined as follows.

$$SCORE1=\sum_{j=1}^{m}\left|C_j'\right|$$

Suppose $O_i \in C_j$, we define $Dintra(O_i)$ and $Dinter(O_i)$ as:

$$Dintra(O_i)=d(O_i,S_j)$$

$$Dinter(O_i)=\min_{1\le k\le m,k\ne j} d(O_i,S_k)$$

where $d(O_i,S_j)$ and $d(O_i,S_k)$ are defined in (1). So *SCORE1* is defined below:

$$SCORE2=\begin{cases}\sum_{i=1}^{n}(Dinter(O_i)*w-Dintra(O_i)) & ,if \quad SCORE1=N\\ 0 & ,if \quad SCORE1<N\end{cases}$$

where w is a weight. If the value of w is small, we emphasize the importance of $Dintra(O_i)$. This tends to produce more clusters and each cluster tends to be compact. If the value of w is chosen to be large, we emphasize the importance of $Dinter(O_i)$. This tends to produce less clusters and each cluster tends to be loose.

2.3. Crossover and mutation

After the calculation of fitness for each string in the population, the reproduction operator is implanted by using a roulette wheel with slots sized according to fitness. In the crossover phase, for each chosen pair of strings, two random numbers are generated to decide which pieces of the strings are to be interchanged. Each random number is an integer in [1,*N*]. For example, if two random numbers are 2 and 5, then position 2 to 5 of this pair of strings is interchanged. For each pair of chosen strings, the crossover operator is done by probability p_c. In the mutation phase, bits of the strings in the population will be chosen with probability p_m. Each chosen bit will be change from 0 to 1 or from 1 to 0.

3. Modeling methodology

3.1. Data collection

We studied a part of a large telecommunications software system. The part we studied consists of about 480 routines that are programmed in C language. Product metrics were collected at the routine level using the DATRIX software analyzer [13], [14]. The following is a list of the product metrics used in this study [15].

Table 1. Product metrics

Symbol	Description
RtnArgXplSum	Sum of the explicit argument numbers (actual parameters) passed to other function by all the explicit function calls made in the routine
RtnCalXplNbr	Number of explicit function calls in the routine
RtnCplCtlSum	Total (sum) complexity of the control predicates (test expressions) composing the decision and loop statements within the routine
RtnStmDecObjNbr	Number of variable/object declaration statements in the routine
RtnStmDecPrmNbr	Number of parameters of the routine
RtnScpNstLvlSum	Sum of nesting level values for all scopes in the routine, as illustrated by the following equation: $ScpNstLvlSum = \sum_{\forall Scope"s"inRoutine} Level(s)$ where $Level(s)$ is the nesting level of a scope s in the routine.
RtnStmCtlNbr	Number of control-flow statements in the routine

Note that one could select other software metrics to characterize the software. The selected metrics illustrate the methodology and its potential as a tool for management, but our methodology is not dependent on this particular set of metrics. A fault was defined as a reported problem that caused a change in the code. Data on faults in each routine were collected in system test. In this study, a routine including a fault is considered as a *fault-prone* module.

3.2. Neural Network Modeling

Many different models of neural networks have been proposed [16]. The feed-forward multi-layer networks with the Backpropagation learning algorithm are the most commonly used in the software quality prediction field. Figure 1 illustrates our neural network architecture configured for *fault-prone* software module prediction.