Neurocomputing 186 (2016) 66–73

Contents lists available at ScienceDirect

Neurocomputing

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

Software reliability prediction via relevance vector regression

Jungang Lou [a,b], Yunliang Jiang [b,*], Qing Shen [b], Zhangguo Shen [b], Zhen Wang [c], Ruiqin Wang [b]

[a] Institute of Cyber-Systems and Control, Zhejiang Univeristy, 310027 Hangzhou, China
[b] School of Information Engineering, Huzhou University, 313000 Huzhou, China
[c] College of Computer Science and Technology, Shanghai University of Electric Power, 200090 Shanghai, China

ARTICLE INFO

Article history:
Received 21 September 2015
Received in revised form
27 November 2015
Accepted 9 December 2015
Communicated by Liang Wang
Available online 6 January 2016

Keywords:
Software reliability model
Relevance vector machine
Mann–Kendall test
Paired *T*-test

ABSTRACT

The aim of software reliability prediction is to estimate future occurrences of software failures to aid in maintenance and replacement. Relevance vector machines (RVMs) are kernel-based learning methods that have been successfully adopted for regression problems. However, they have not been widely explored for use in reliability applications. This study employs a RVM-based model for software reliability prediction so as to capture the inner correlation between software failure time data and the nearest m failure time data. We present a comparative analysis in order to evaluate the RVMs effectiveness in forecasting time-to-failure for software products. In addition, we use the Mann-Kendall test method to explore the trend of predictive accuracy as m varies. The reasonable value range of m is achieved through paired T-tests in 10 frequently used failure datasets from real software projects.

1. Introduction

In the modern world, computers are used for many different applications, and research on software reliability has become increasingly essential. Software reliability describes the probability that software will operate without failure under given environmental conditions during a specified period of time [1]. To date, software reliability models are among the most important tools in software reliability assessment [2]. Most existing software reliability models, known as parametric models, depend on priori assumptions about software development environments, the nature of software failures, and the probability of individual failures occurring. Parametric models may exhibit different predictive capabilities across different software projects [3–8], and researchers have found it almost impossible to develop a parametric model that can provide accurate predictions under all circumstances. To address this problem, several alternative solutions have been introduced over the last decade. One possible solution is to employ artificial neural networks (ANNs) [9–17]. Karunanithi et al., Dohi et al., Cai et al., Ho et al., Tian and Noore and Hu et al. used both classical and recurrent multi-layer perceptron neural networks to forecast software reliability. ANNs have proven to be universal approximates for any nonlinear continuous function with an arbitrary accuracy. Consequently, they represent an alternative method in software reliability modeling and predicting. Unlike traditional statistical models, ANNs are data-driven, nonparametric weak models [9,11,13,14]. ANN-based software reliability models require only failure history as an input, and they can predict future failures more accurately than some commonly used parametric models. However, ANNs suffer from a number of weaknesses, including the need for numerous controlling parameters, difficulty in obtaining a stable solution, and a tendency to cause over fitting. A novel type of learning machine, kernel machines (KMs), is emerging as a powerful modeling tool, and it has received increasing attention in the domain of software reliability prediction. Kernel-based models can achieve better predictive accuracy and generalization performance, thus arousing the interest of many researchers [18–22]. Generally speaking, KMs have been successfully applied to regressions, with remarkable training results even given a relatively small dataset $D=\{(x_1,y_1),(x_2,y_2),\ldots,(x_l,y_l)\}\in R^d\times R$, where x_t are input vectors, y_t are output vectors, $t=1,2,\ldots,l$, d is a dimension of x_t, and l is the number of observed input/output pairs [18].

Examples of KMs include support vector machines (SVMs) and relevance vector machines (RVMs). Vapnik [18] developed SVMs with the goal of minimizing the upper boundary of the generalization error consisting of the sum of the training error and confidence interval, which appears to be less computationally demanding. Tian and Noore [19] proposed an SVM-based model for software reliability prediction that embraces some remarkable characteristics of SVMs, including good generalization performance, absence of local minima, and sparse solution representation. Pai and Hong [20] and Yang and Li [21] also made efforts to develop SVM-based reliability models, showing that

* Corresponding author.
E-mail addresses: loujungang0210@hotmail.com (J. Lou), jylsy@hutc.zj.cn (Y. Jiang), sq@hutc.zj.cn (Q. Shen), szgxx@hutc.zj.cn (Z. Shen), wangzhenqq@hotmail.com (Z. Wang), angelwrq@163.com (R. Wang).

http://dx.doi.org/10.1016/j.neucom.2015.12.077
0925-2312/

these models can achieve good prediction accuracy. However, SVMs are sensitive to uncertainties because of the lack of probabilistic outputs, as well as the need to determine a regularization parameter and select appropriate kernel functions to obtain optimal prediction accuracy [22–25].

This paper proposes a new data-driven approach for predicting software reliability using RVM [23–27] to capture the uncertainties in software failure data and predictions about possible present and future aquifer conditions. RVM adopts kernel functions to project the input variables into a high-dimensional feature space, in order to extract the latent information. Compared to SVM, it uses fewer kernel functions and avoids the use of free parameters [28–30].

The kernel-based software reliability modeling process also focuses on choosing the number of past observations related to the future value. Some researchers suggest that failure behavior earlier in the testing process has less impact on later failures, and therefore not all available failure data should be used in model training. However, to the best of our knowledge, such claims lack either theoretical support or experimental evidence. This study uses the Mann–Kendall test and paired *T*-test [31–33] to investigate the appropriate number of past observations related to the future value for RVM-based software reliability modeling.

The paper is organized as follows. After explaining the background of the research, Part 2 outlines the principle of RVM for regression. Part 3 introduces the framework for software reliability prediction based on RVM and describes how RVM regression can be used in predicting software failure time. Part 4 discusses the process for RVM-based software reliability models and presents experimental datasets and measures for evaluating predictability. Following that, Part 5 explains the Mann–Kendall test and paired *T*-test, demonstrates the detailed experimentation process, and analyzes the experimental results on the 10 datasets. Finally, Part 6 concludes the paper.

2. RVM for regression

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. The goal of SVMs classification is to separate an n-dimensional data space (transformed using nonlinear kernels) by an $(n-l)$-dimensional hyper-plane that creates the maximum separation (margin) between two classes. This technique can be extended to regression problems in the form of support vector regression. Regression is essentially an inverse classification problem where, instead of searching for a maximum margin classifier, a minimum margin fit needs to be found. However, SVMs are not well suited to software reliability prediction due to the lack of probabilistic outputs. Tipping [24–27] introduced the RVM which makes probabilistic predictions and yet which retains the excellent predictive performance of the support vector machine. It also preserves the sparseness property of the SVM. The RVM is a Bayesian form representing a generalized linear model of identical functional form to the SVM, and it is a Bayesian sparse kernel technique for regression, which introduces a prior over the model weights dominated by a set of hyper-parameters, whose most probable values are iteratively estimated from the data. In addition to the probabilistic interpretation of its output, it uses far fewer kernel functions for comparable performance. We give a brief review of RVM for regression. For a more detailed discussion on RVM, readers can refer to [24–27]. Assuming that a total of N pairs of training patterns are given during RVM learning process,

$$(x_1, t_1), (x_2, t_2), \ldots, (x_i, t_i), \ldots, (x_N, t_N),$$

where the inputs are n-dimensional vectors $x_i \in R^n$ and the target outputs are continuous values $t_i \in R$. The RVM model used for function approximation is:

$$t = y(x; w) = \sum_{i=1}^{M} w_i K(x, x_i) + w_0 \tag{1}$$

where these $\{w_i\}$ are the parameters of the model, generally called weights, and $K(,)$ is the kernel function. Assuming that each example from the data set has been generated independently (an often realistic assumption, although not always true), the likelihood of all the data is given by the product:

$$p(t|\sigma^2) = \prod_{i=1}^{n} N(t_i|y(x_i; w), \sigma^2) = (2\pi\sigma^2)^{N/2} \exp\left(-\frac{\|t - \Phi w\|^2}{2\sigma^2}\right)$$

where $w = [w_0, w_1, w_2, \ldots, w_N]^T$, $\Phi = [\phi(x_1), \phi(x_2), \ldots, \phi(x_N)]^T$, and $\phi(x_n) = [1, K(x_n, x_1), K(x_n, x_2), \ldots, K(x_n, x_N)]^T$.

Next we introduce a prior distribution over the parameter vector w. The key difference in the RVM is that we introduce a separate hyperparameter α_i for each of the weight parameters w_i instead of a single shared hyperparameter. Thus, the weight prior takes the form:

$$p(w|\alpha) = \prod_{i=0}^{N} \frac{\alpha_i}{\sqrt{2\pi}} exp\left(-\frac{\alpha_i w_i^2}{2}\right), \quad \alpha = [\alpha_1, \alpha_2, \ldots, \alpha_N].$$

Having defined the prior, Bayesian inference proceeds by computing, from Bayes rule, the posterior over all unknowns give the data:

$$p(w, \alpha, \sigma^2|t) = \frac{p(t|w, \alpha, \sigma^2)p(w, \alpha, \sigma^2)}{p(t)}. \tag{2}$$

Then, given a new test points x_*, predictions are made for the corresponding target t_*, in terms of the predictive distribution:

$$p(t_*|t) = \int p(t_*|w, \alpha, \sigma^2)p(w, \alpha, \sigma^2|t)\, dw\, d\alpha\, d\sigma^2 \tag{3}$$

We cant compute the posterior $p(w, \alpha, \sigma^2|t)$ in (1) directly. Instead, we decompose the posterior as:

$$p(w, \alpha, \sigma^2|t) = p(w|t, \alpha, \sigma^2)p(\alpha, \sigma^2|t).$$

The posterior distribution over the weights is thus given by:

$$p(w|t, \alpha, \sigma^2) = \frac{p(w, \alpha, \sigma^2|t)}{p(\alpha, \sigma^2|t)} = \frac{p(t|w, \sigma^2)p(w|\alpha)}{p(t|\alpha, \sigma^2)} = \frac{p(t|w, \sigma^2)p(w|\alpha)}{\int p(t|w, \sigma^2)p(w|\alpha)\, dw}$$
$$= (2\pi)^{-\frac{N+1}{2}} |\Sigma|^{-(1/2)} * \exp\left\{-\frac{(w-\mu)^T \Sigma^{-1} (w-\mu)}{2}\right\},$$

where the posterior covariance and the mean are respectively:

$$\mu = \sigma^{-2} \Sigma \Phi^T t,$$
$$\Sigma = (A + \sigma^{-2} \Phi^T \Phi)^{-1},$$
$$A = diag(\alpha_0, \alpha_1, \ldots, \alpha_N).$$

By integrating the weights, we obtain the marginal likelihood for the hyper parameter:

$$p(t|\alpha, \sigma^2) = (2\pi)^{-(N/2)} |\Omega|^{-(1/2)} \exp\left\{-\frac{t^T \Omega^{-1} t}{2}\right\}, \tag{4}$$

where $\Omega = \sigma^2 I + \Phi A^{-1} \Phi^T$. Our goal is now to maximize (4) with respect to the hyper parameters α, σ^2. We simply set the required derivatives of the marginal likelihood to zero and obtain the following re-estimation equations:

$$\alpha_i^{new} = \frac{\gamma_i}{\mu_i^2},$$
$$(\sigma^2)^{new} = \frac{\|t - \Phi\mu\|^2}{N - \sum_{i=0}^{N} \gamma_i},$$
$$\gamma_i = 1 - \alpha_i \sum_{ii}.$$