Bayesian Software Reliability Prediction
Using Software Metrics Information

M. P. Wiper, A. P. Palacios and J. M. Marín
Department of Statistics, Carlos III University of Madrid, Spain
(Received October 2010, accepted March 2011)

Abstract: This paper analyzes software reliability models where covariate information in the form of software metrics is available. Our approach uses neural network regression to estimate failure rates in models based on inter failure times or numbers of failures. Inference is carried out using a Bayesian approach which is implemented using the free software packages Winbugs and R. A real data example is used to illustrate the paper.

Keywords: Bayesian analysis, neural networks regression, software reliability.

1. Introduction

Software reliability is usually defined as the probability that a piece of software runs without failing under certain operational conditions for a given time, see [16]. Software testing is concerned with the development of different testing strategies for the detection and correction of faults in order to improve reliability. Various approaches to software testing and software reliability measurement have been developed.

Early stage software testing often focuses on randomly generating inputs to the software from the operational profile, in order that the maximum number of faults in the software can be detected and corrected, see [9]. An indirect approach to the measurement of software quality is via the use of software metrics, that is characteristics of a piece of software such as the number of lines of code or measures of effort such as the number of man hours of work on the software, see [5, 6]. These characteristics can then be related to a measure of reliability of the software using e.g. regression, see [27], neural networks, see [11], Bayesian networks, see [1] and other approaches, see [3].

The later stages of the software development process are concerned with prediction of software reliability and with estimation of when the software is sufficiently reliable to be released. Here, software reliability models are typically applied. Such models assume that the software is run under the operational profile either until a failure is observed, when the fault causing the failure is possibly imperfectly corrected and the software is then run until the next failure and so on, or for a fixed time period, when the number of failures in this period are recorded and the software is then, possibly imperfectly debugged, then for another fixed time period and so on. These different types of models are classified as type I and type II models by [20].

Starting from [17] in the case of type I models and [28] in the case of type II models, a large number of software reliability models have been developed. A good review is given in [28]. Both classical and Bayesian approaches to fitting such models have been developed. In
this article, we consider a Bayesian approach to reliability estimation. Good reviews of Bayesian methods in software reliability which emphasize the advantages of the Bayesian philosophy as compared to classical approaches, are provided by [13, 21, 25].

Most software reliability models developed thus far do not consider the use of covariate information. However, in many cases, software metrics information which can be used to aid in the prediction of software reliability will be available and a few approaches to using such information have been developed. In particular, [26] consider how to incorporate software metrics data into the Jelinski Moranda type I model. [18] develops a regression based approach to type II models using time varying metrics and [19] develops an approach to type II models based on proportional hazards type ideas.

In this paper, we develop a unified approach to type I and type II software reliability models in the presence of metrics information based on the use of Bayesian nonparametric regression via neural networks. Neural network based approaches are not new in software reliability prediction, but, in general such networks have been developed as direct predictors of failures or inter failure times. However, it has been suggested that such models are often prone to overfitting and are not very good at out of sample prediction of reliability. See [2]. In contrast, our approach is based on using a simple parametric model for failure times or numbers of failures where the failure rate is modeled nonparametrically.

The paper is organized as follows. In Section 2 we introduce the type I and type II neural network regression models for software reliability and then in Section 3, we indicate how to carry out Bayesian inference for these models using the software package WinBugs outlined in [15]. Then in Section 4, we illustrate our approach with some real data examples and finally, in Section 5, we give some conclusions and possible extensions of this work.

### 2. Software Reliability Models Using Covariate in Formation

Consider type I software reliability models where the times between successive software failures, say $T_1, T_2, \ldots$ are observed and where it is presumed that the software is corrected, possibly imperfectly, after each failure. Then, it is natural to assume a nonhomogeneous Poisson process for failures so that we model

$$T_i | \lambda_i \sim \text{Ex}(\lambda_i), \quad (1)$$

for $i = 1, 2, \ldots$. Many standard software reliability models assume this basic exponential form. For instance, the Jelinski Moranda model sets

$$\lambda_i = (N - i + 1) \mu, \quad (2)$$

where $N$ represents the number of faults in the original code, $\mu$ is the fault discovery rate and perfect fault correction is assumed.

Here, we suppose that after each software failure is observed, the code is modified and software metrics reflecting the state of the code are evaluated. Then, we relate the failure rate of the software to the software metrics as follows:

$$\log \lambda_i = g(x_i), \quad (3)$$

where $x_i = (x_{i1}, \ldots, x_{ip})^T$ are the covariates available after $i - 1$ failures have been observed. One possibility would now be to consider a linear model for the function $g(\cdot)$ which implies a standard exponential regression model for the inter failure times, see [18]. However, in
Bayesian Software Reliability Prediction Using Software Metrics Information

In many cases, the relation between the log failure rate and the metrics may be highly non-linear and therefore, a nonparametric model should be preferred. Here we use a feed forward neural network, that is:

\[
g(x) = \beta_0 + \sum_{j=1}^{k} \beta_j \gamma_j^T x \quad \text{where} \\
\gamma(c) = (1 + \exp(-c))^{-1}
\]  

and \( \gamma_j = (\gamma_{j1}, \ldots, \gamma_{jp})^T \).

A neural network can be thought of as an information processing unit designed to mimic the way the brain works, as illustrated in Figure 1. The network contains \( k \) neurons, each of which has \( p \) inputs and a single output and an activation function, \( \Psi \) for limiting the output to the unit interval. The final output of the network is then a weighted average of the individual neuron outputs. For a good introduction in the software reliability context, see [2] and for applications in fault detection using software metrics, see [12].

![Figure 1. Neural network representation of the growth function model.](image)

Type II models can be treated in a similar way. In this case, we suppose that in time period \( i \), then software metrics are available and that the number of faults, \( Y_i \), in the next time period are modeled according to a Poisson distribution as

\[
T_i | \lambda_i \sim \text{Po}(\lambda_i),
\]

where the failure rate \( \lambda_i \) follows a neural network model as in (4)-(5). At the end of each time period, it is assumed that the software is corrected and the metrics are recalculated as earlier.

Finally, we should note that for ease of implementation, as is typical with the neural network models, from now on, we will assume throughout that the metrics, \( x \), are normalized to take values between zero and one, see [30].
3. Bayesian Inference

In order to undertake Bayesian inference for the models described in Section 2, prior distributions for the neural network parameters $\beta$ and $\gamma$ must be defined. Typically in neural network models, there is little prior information and therefore, relatively diffuse prior distributions are assumed. Here we set $\beta_0 \sim \text{N}(0,10^6)$ and use a hierarchical prior structure for the remaining parameters, that is:

$$\beta | m_\beta, \tau_\beta \sim \text{N}\left(m_\beta, \frac{1}{\tau_\beta}\right),$$

$$m_\beta \sim \text{N}(0,10^6),$$

$$\tau_\beta \sim \text{Ga}(0.001, 0.001),$$

$$\gamma | m_\gamma, \tau_\gamma \sim \text{N}\left(m_\gamma, \frac{1}{\tau_\gamma}\right),$$

$$m_\gamma \sim \text{N}(0,10^6),$$

$$\tau_\gamma \sim \text{Ga}(0.001, 0.001).$$

Given this prior structure, a closed form for the posterior parameter distribution is not available. The standard approach in such situations is then to apply Markov chain Monte Carlo or MCMC algorithms to generate an approximate sample from the posterior distribution, see [8] for a good review. Many different MCMC algorithms have been proposed for sampling from neural network models, see [14], and in general, the efficiency of such samplers is highly model dependent. Also there are often many difficulties in fitting neural network models and, in the case of both classical and Bayesian inference, issues such as data normalization and the learning algorithms used can be highly critical for the performance of the model. See [30] for a good review.

Figure 2. WinBugs style representation of type II model.
Here however, we prefer to use a simple algorithm based on the use of a Gibbs sampler as implemented in WinBugs, see [15]. This approach is an automatic sampler which is based on the hierarchical structure of the model as illustrated in Figure 2. In this figure, the oval nodes represent parameters or dependent variables, the rectangular nodes represent independent variables or known values, the arrows show the dependence structure and the double arrows indicate a logical relation, so that for example, \( \lambda(i) \) is a function of \( x_i \) and the neural network parameters.

Given this hierarchical model structure, and the data and a set of initial values for the MCMC sampler, a simple WinBugs code can be written to generate a sample from the posterior parameter distribution as below:

```plaintext
model{
  for (j in 1:k){
    beta[j] ~ dnorm(mubeta,taubeta)
    for (r in 1 : p){
      gamma[j,r] ~ dnorm(mugamma,taugamma)
    }
  }
  for (i in 1 : n){
    y[i] ~ dpois(lambda[i])
    for (j in 1 : k){
      for (r in 1 : p){
        gpt[i,j,r] <- gamma[j,r]*x[i,r]
      }
      psipt[i,j] <- beta[j]/(1+exp(-sum(gpt[i,j,])))
    }
    log(lambda[i]) <- beta0 + sum(psipt[i,])
  }
  mubeta ~ dnorm(0.0,0.001)
  taubeta  ~ dgamma(0.001,0.001)
  mugamma ~ dnorm(0.0,0.001)
  taugamma ~ dgamma(0.001,0.001)
  beta0 ~ dnorm(0.0,0.001)
}
```

WinBugs can be run directly, or as here via the interface R2WinBugs within the package R.

Then, given a sample of values, say \((\beta^{(l)}, \gamma^{(l)})\) for \(l = 1, \ldots, L\), from the posterior parameter distribution, then, for example the mean number of failures for a the software given metrics values \(x\) can be estimated by

\[
E[\lambda|x,y] \approx \frac{1}{L} \sum_{l=1}^{L} \lambda^{(l)} \quad \text{where} \quad \log \lambda^{(l)} = \left( \beta_0^{(l)} + \sum_{k=1}^{K} \beta_k^{(l)} \Psi(y^{(l)^T} x) \right).
\]

The predictive distribution for the number of failures given metrics values \(x\) can be estimated in a similar way as

\[
P(Y = y | \text{data, } x) \approx \frac{1}{L} \sum_{l=1}^{L} \frac{\lambda^{(l)y} e^{-\lambda^{(l)}}}{y!}.
\]

### 3.1 Model Selection

An important aspect of neural network based models is model selection. Both the number of nodes of the neural network and the regressors or independent variables to be
included in the model must be considered. For classical model selection, criteria such as the Akaike (AIC) or Bayesian (BIC) information criteria which weight the log likelihood of the data according to the number of parameters in the model are usually employed. The usual Bayesian equivalent to the classical Akaike information criterion is the deviance information criterion or DIC developed in [22]. This criterion can be calculated directly within WinBugs. However, in the context of neural networks, calculation of this criterion is unstable, due to the lack of identifiability of the model. Therefore, we prefer to use a variant of the DIC, denoted DIC3 in [4]. This criterion is defined, for the type II model with data \( y \) and model \( M \) as

\[
-4E[ \log f(y \theta) | y, M ] + 2\log \hat{f}(y | y, M),
\]

where

\[
\hat{f}(y | y, M) = \prod_i \hat{f}(y_i | y, M),
\]

and

\[
\hat{f}(y_i | y, M) = \frac{1}{L} \sum_{l=1}^L f(y_i | y, \lambda_l^{(i)}, M) = \frac{1}{L} \sum_{l=1}^L \frac{\lambda_l^{(i)} e^{-\lambda_l^{(i)}}}{y_i!},
\]

and can be easily calculated from the output of WinBugs. As with the AIC and BIC, lower values of this criterion imply better fitting models. For a full review, see [4].

4. Example

In [18], two data sets from a software development for real time command and control systems are presented. Data set 1 (2) consists of failure numbers during 17 (14) weeks with metrics information on week \( x_1 \), weekly execution time \( x_2 \), CPU hr, failure identification work \( x_3 \) person hr and computer time failure identification \( x_4 \) CPU hr. Here we use data set 1 as a training data set and then use the posterior distribution derived from this data set to make predictions for data set 2.

The WinBugs sampler was run for 100000 iterations to burn in and 100000 iterations assuming equilibrium. Table 1 shows the estimated values of the DIC3 criterion from fitting various different models to data set 1. The first two models are neural networks regression models and the last two models are a constant model and a linear regression model for \( g \). The neural network models are seen to outperform the constant and linear models in terms of the DIC3 criterion. Note that many other neural and linear regression models were also applied but performed worse than the optimal models presented here.

<table>
<thead>
<tr>
<th>Number</th>
<th>Model for ( g )</th>
<th>DIC3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \sum_{j=1}^2 \beta_j \Psi(x_1, x_2, x_4) )</td>
<td>64.51</td>
</tr>
<tr>
<td>2</td>
<td>( \beta_0 + \sum_{j=1}^3 \beta_j \Psi(x_1, x_2, x_3, x_4) )</td>
<td>65.87</td>
</tr>
<tr>
<td>3</td>
<td>( \beta_0 )</td>
<td>87.42</td>
</tr>
<tr>
<td>4</td>
<td>( \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 )</td>
<td>68.38</td>
</tr>
</tbody>
</table>
The optimal model according to the DIC3 criterion is model 1, that is a neural network with regressors $x_1, x_2, x_4$, two nodes and no constant term. Model 2, that is a neural network with a constant term and regressors $x_1, x_2, x_3, x_4$ has an insignificantly different DIC3 value. The DIC of the linear regression model 4 is only slightly higher, which suggests that all three of these models provide reasonable fits here. However the constant model 3 has a much higher DIC which suggests that this model can be rejected.

As well as goodness of fit, it is also important to consider the predictive capacity of these models. Figure 3 shows the out of sample predicted mean number of failures and 95% credible intervals for models 1, 2 and 4 for data set 2.

![Figure 3. Observed values (circles), one step ahead predictive means (asterisks) and 95% predictive intervals for models 1 (solid line), 2 (dashed line) and 4 (dotted line).](image)

Models 1 and 2 appear to predict the unobserved data very well and in all cases, the observed values fall within the 95% predictive intervals. In week 2 however, the observed value falls outside the 95% predictive interval for the regression model 3. Thus, the predictions for all 3 models again seem reasonable. Predictive mean squared errors (MSEs), $1/n\sum_{i=1}^{n}(y_i - E[y_i | y_1, \ldots, y_{i-1}])^2$ were also calculated for all three models. In this case, the MSEs were 4.43 for model 1, 3.44 for model 2 and 4.52 for model 4. Thus, according to this criterion, both neural network based models outperform the regression model and, in particular, the full neural network model appears to provide the best predictions. Under other criteria such as the posterior predictive loss performance of [7], model 2 is also selected as the best predictor.

5. Conclusions and Extensions

In this paper, we have presented a unified approach to software reliability prediction in the presence of software metrics information. It has been shown that Bayesian inference for
neural network regression models in this context can be implemented using the WinBugs software package and that model choice can be undertaken using the DIC3 criterion.

The main advantage of using WinBugs in implement the Bayesian approach developed here lies in the simplicity and rapidity of the procedure. However, one disadvantage is that the initial values for running the WinBugs program have to be quite carefully chosen as sometimes, WinBugs can hang, or occasionally, problems with the numerical routines in WinBugs can be observed. This suggests that it may be preferable in some cases to use other more specific approaches to implement the neural network regression models.

A number of extensions are possible. Firstly, it would be interesting to analyse the predictive performance of these neural network models on larger data sets and also on type I data sets. Secondly, when software metrics are not available at all time periods, then the approach can be extended by placing a prior distribution on the software metrics values and then using the predictive distributions for the missing data to enable prediction of the software failures. This can be handled in a relatively straightforward manner in WinBugs. Thirdly, alternative models for the function \( g() \) could be considered. One possibility is to use spline functions, see [23] and a second approach would be to apply Gaussian processes, that is Bayesian support vector machines, see [24]. A fourth possibility is to incorporate nonparametric aspects within parametric software reliability models. Thus, for example the Jelinski Moranda model in (2) could be modified by allowing the rate, \( \mu \), or the fault number, \( N \), to depend upon the software metrics information available. Finally, a fourth extension is to incorporate these techniques into sequential software testing procedures as studied in [29]. Work is in progress on such models.

Acknowledgements

The authors acknowledge support provided by the Spanish Ministry of Science and Education and from the Community of Madrid. Thanks are also due to two referees and the editors for their helpful comments on an earlier version of this manuscript.

References


Authors’ Biographies:

**M. P. Wiper** is an Assistant Professor of Statistics at the Carlos III University of Madrid. His interests are in Bayesian inference for stochastic processes and software reliability systems. He is author of several book chapters and journal publications in the Bayesian statistics and software reliability literature.

**A. P. Palacios** is a teaching assistant in Statistics at Carlos III University in Madrid, Spain. Her research interests are in the areas of Bayesian statistics, non-parametric inference, software reliability and biostatistics. She has a doctoral fellowship under the supervision of Michael P. Wiper and Juan Miguel Marín.

**J. M. Marin** is an Assistant Professor of Statistics at the Carlos III University of Madrid. His interests are in the areas of Bayesian data analysis, Bioinformatics and statistical computing. He is co-author of several book chapters and publications in Bayesian statistics and biostatistics literature.