Ensemble Learning-based Methods for Software Reliability Prediction

呉　敬馳、鄭　俊俊、土肥　正、岡村寛之
広島大学大学院先進理工系科学研究科 *
Jingchi Wu, Junjun Zheng, Tadashi Dohi, Hiroyuki Okamura
Graduate School of Advanced Science and Engineering
Hiroshima University

1 Introduction

Software reliability is important for software project managers and software test engineers, since it provides an effective reference to determine the release time of software system. Quantitative software reliability, defined as the probability that software failures caused by software faults do not occur over a period of time, is the measure used for this purpose. In the past literatures, there are many software reliability models (SRMs) based on different stochastic processes have been proposed. Among the hundreds of existing SRMs, non-homogeneous Poisson process (NHPP) based SRMs received the considerable attentions [1,2,8,9,14,15,17,21,22]. Okamura and Dohi [16] developed Software Reliability Assessment Tool on Spreadsheet (SRATS) which implements the above 11 representative NHPP-based SRMs.

In general, it is necessary to select a single SRM, e.g. one of the 11 kinds of representative NHPP-based SRMs in SRATS, with the best goodness-of-fit performance under the observed software fault count data, using any information criteria such as Akaike Information Criterion (AIC) [3] for software reliability prediction. However, Wu et al. [20] showed that AIC, BIC [18] or others [4, 19] merely select the single model with the best goodness-of-fit performance to the underlying software faults data, without guaranteeing that the selected model always has the best predictive performance. To overcome the above disadvantage, Lyu and Nikora [12] proposed an equally-weighted linear combination model, which is a combination model based on three component SRMs. The original idea to use such a linear weighted combination model as

〒739-8527 東広島市鏡山 1-4-1

a meta-predictor by taking a linear weighted combination of two or more predictions came from Abdel-Ghaly et al. [1], where the model weights have to be chosen in some optimal way.

Unfortunately, it should be noted that the determination of the model weights in [12] is rather heuristic and problematic. First, the determination of the model weights was completely irrelevant to the principle of parameter estimation, so the frequentist approach and the Bayesian approach were confused in the component SRMs. Second, the optimal or statistically meaningful model weights were not chosen at all. In this paper, we propose refined software reliability prediction method with the adaboosting-based model averaging techniques. Recently, the model averaging has gained much popularity in prediction of regression-based models (see e.g. [6]).

The paper is organized as follows. Section 2 summarizes the 11 representative NHPP-based SRMs and introduces the AIC-based single model approach for software reliability prediction. In Section 3, we first summarize the existing linear weighted combination model with adaboosting. Then, we propose an alternative deterministic adaboosting-based approch for software reliability prediction. Section 4 is devoted to numerical experiments for investigating the predictive performances of our refined adaboosting-based software reliability prediction method. Finally, the paper is concluded with some remarks in Section 5.

2 NHPP-based SRMs

2.1 Definition

Let $N(t)$ represent the cumulative number of software faults detected in a system testing by time t (≥ 0). Suppose that $N(t)$ is described by a non-homogeneous Poisson process (NHPP). In this case, it holds that

$$Pr\left\{N(t)=n\right\} = \frac{\{\Lambda(t)\}^n\, e^{-\Lambda(t)}}{n!}, \tag{1}$$

where $\Lambda(t) = \int_0^t \lambda(x)dx = E[N(t)]$ is called the mean value function and is the expectation of $N(t)$, and $\lambda(t)$ (≥ 0) is called the intensity function. Therefore, the NHPP-based SRM is uniquely determined by $\Lambda(t)$ or $\lambda(t)$. In the past literatures, hundreds of $\Lambda(t)$ have been proposed. Table 1 represents the 11 kinds of representative NHPP-based SRMs implemented in SRATS [16]. Then, the quantitative software reliability $R(t_u, t_s)$ between time interval (t_u, t_s) is defined by

$$R(t_u, t_s) = e^{\Lambda(t_s)-\Lambda(t_u)}, \tag{2}$$

which means the probability that software faults are not detected during the time interval $[t_u, t_s)$.

2.2 Maximum Likelihood Estimation

Next, we briefly introduce the parameter estimation. Suppose that we observed n time-domain data, $\boldsymbol{t} = \{t_1, t_2, \ldots, t_n\}$, where t_i denotes the i-th software fault-detection time. $\boldsymbol{\theta}$ is the model

Table 1. 11 kinds of representative NHPP-based SRMs.

Mean Value Function	$\Lambda(t;\boldsymbol{\theta})$ $(\boldsymbol{\theta}=a,b,c>0)$
Exp [8]	$\Lambda(t;\boldsymbol{\theta})=a(1-e^{-bt})$
Gamma [21, 22]	$\Lambda(t;\boldsymbol{\theta})=a\int_0^t \frac{c^b s^{b-1}e^{-cs}}{\Gamma(b)}ds$
Pareto [1]	$\Lambda(t;\boldsymbol{\theta})=a\left(1-\left(\frac{c}{t+c}\right)^b\right)$
TruncNormal [17]	$\Lambda(t;\boldsymbol{\theta})=a\frac{F(t)-F(0)}{1-F(0)}$, $F(t)=\frac{1}{\sqrt{2\pi}b}\int_t^{-\infty} e^{-\frac{(s-c)^2}{2b^2}}ds$
LogNormal [2, 17]	$\Lambda(t;\boldsymbol{\theta})=a\frac{1}{\sqrt{2\pi}b}\int_{\log(t)}^{-\infty} e^{-\frac{(s-c)^2}{2b^2}}ds$
TruncLogist [14]	$\Lambda(t;\boldsymbol{\theta})=a\frac{F(t)-F(0)}{1-F(0)}$, $F(t)=\frac{1}{1+e^{-\frac{t-c}{b}}}$
LogLogist [9]	$\Lambda(t;\boldsymbol{\theta})=a\frac{1}{1+e^{-\frac{\log(t)-c}{b}}}$
TruncEVMax [15]	$\Lambda(t;\boldsymbol{\theta})=a\frac{F(t)-F(0)}{1-F(0)}$, $F(t)=e^{-e^{-\frac{t-c}{b}}}$
LogEVMax [15]	$\Lambda(t;\boldsymbol{\theta})=ae^{-e^{-\frac{\log(t)-c}{b}}}$
TruncEVMin [15]	$\Lambda(t;\boldsymbol{\theta})=a\frac{F(t)-F(0)}{1-F(0)}$, $F(t)=e^{-e^{-\frac{t-c}{b}}}$
LogEVMin [15]	$\Lambda(t;\boldsymbol{\theta})=ae^{-e^{-\frac{\log(t)-c}{b}}}$

parameter vector of the mean value function $\Lambda(t)$ and intensity function $\lambda(t)$. The likelihood function is given by

$$\mathcal{L}(\boldsymbol{\theta};\boldsymbol{t})=e^{-\Lambda(t_n;\boldsymbol{\theta})}\prod_{i=1}^{n}\lambda(t_i;\boldsymbol{\theta}). \tag{3}$$

Taking the logarithm of Eq.(3), we obtain the log likelihood function as

$$\ln\mathcal{L}(\boldsymbol{\theta};\boldsymbol{t})=\sum_{i=1}^{n}\ln\lambda(t_i;\boldsymbol{\theta})-\Lambda(t_n;\boldsymbol{\theta}) \tag{4}$$

under the failure truncation at time t_n. Then, the maximum likelihood estimate (MLE), $\tilde{\boldsymbol{\theta}}$, of model parameters is defined by

$$\tilde{\boldsymbol{\theta}}=\arg\max_{\boldsymbol{\theta}}\ln\mathcal{L}(\boldsymbol{\theta};\boldsymbol{t}). \tag{5}$$

2.3 Single Model-based Software Reliability Prediction

Assume that there are M kinds of candidate SRMs $\{\Lambda_i(t)\}_{i=1,2,\ldots,M}$, e. g. $M=11$ in Table 1. In general, practitioners select only one model among M candidate models in software reliability

prediction. When we obtain the MLEs of each candidate models, the goodness-of-fit performance of each model can be measured by the AIC :

$$\mathrm{AIC} = -2 \cdot \ln \mathcal{L}(\boldsymbol{\theta}) + 2 \cdot k, \tag{6}$$

where k is the number of model parameters in the NHPP-based SRMs. The smaller AIC denotes the better goodness-of-fit performance. The AIC-based single model approach for software reliability prediction is using the minimum AIC model among M candidate models for prediction.

3 Ensemble Learning-based Software Reliability Prediction

3.1 Linearly Weighted Combination Model

Linearly weighted combination model (LWCM) is the most natural combination model. Suppose that there are M NHPP-based SRMs with M kinds of software fault-detection time distributions (*e.g.* $M = 11$ in Table 1). Let k $(= 1, 2, \ldots, M)$ be the component model indicator, i.e., we rewrite the mean value functions and their associated model parameters by $\Lambda_k(t; \boldsymbol{\theta}_k)$ $(k = 1, 2, \ldots, M)$. Let w_k be the model weights. Then, LWCM is given by

$$\tilde{\Lambda}(t) = \tilde{\Lambda}(t; \omega_k, \tilde{\boldsymbol{\theta}}_k, k = 1, 2, \ldots, M) = \sum_{k=1}^{M} \omega_k \Lambda_k(t; \tilde{\boldsymbol{\theta}}_k), \tag{7}$$

where $\omega_k, \tilde{\boldsymbol{\theta}}_k$ are estimated by means of maximum likelihood estimation. It should be noted that LCWM has a large number of parameters for parameter estimation, which seems to be significantly more challenging, compared to the single model approach in Subsection 2.3.

3.2 AdaBoosting-based Multi-combination Model

Li et al. [10] proposed an adaboosting-based multi-combination model (AMCM) for software reliability prediction. Suppose that n time-domain data, $\boldsymbol{t} = \{t_1, t_2, \ldots, t_n\}$, are observed, where t_i denotes the i-th software fault-detection time. We resample the underlying data, where $\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_P$ are P set of resapmled data, where P is need to be deterministic in advance empirically. AMCM can be desribed as

$$\tilde{\Lambda}(t) = \sum_{p=1}^{P} \omega_p \Lambda_p(t; \tilde{\boldsymbol{\theta}}_p), \tag{8}$$

where $\Lambda_p(\cdot)$ is mean value function of the minimum AIC model among M candidate models under the resampled data $\boldsymbol{t}_p$, $\tilde{\boldsymbol{\theta}}_p$ is estimated with the resampled data $\boldsymbol{t}_p$. Resampled data $\boldsymbol{t}_p$ is not generated by simple random sampling but affected by $\boldsymbol{t}_{p-1}$. For more details on the resampled data $\boldsymbol{t}_p$ and model weights ω_p, see [10]. It should be noted that AMCM is just a combination model, and does not implement the adaboosting.R2 algorithm.

3.3 Deterministic Adaboosting Model

Freund and Schapire [7] first proposed a statistical classifier, called *AdaBoosting*, which is implemented as a weighted linear combination of multiple classifiers, called weak learners. Drucker [5] applied the AdaBoosting technique on regression problem and developed a so-called AdaBoosting.R2 algorithm. Here, we apply the AdaBoosting.R2 in the sense of Drucker [5] and propose a deterministic AdaBoosting-based SRM, called the deterministic AdaBoosting-based model (DAM). The DAM consists of two steps; training step and inference step. For the details on DAM, please refer to Algorithm 1 and Algorithm 2.

4 Numerical Experiments

Table 2. Software fault detection time data.

Data Set	No. Fault	Testing Time (CPU Time)	Source	Nature of System
DS1	54	108708	SYS2 [13]	Real time command and control system
DS2	38	67362	SYS3 [13]	Real time command and control system
DS3	136	88682	SYS1 [13]	Real time command and control system
DS4	53	52422	SYS4 [13]	Real time command and control system
DS5	73	5090	Project J5 [11]	Real time command and control system
DS6	38	233700	S10 [13]	Real time command and control system
DS7	41	4312598	S27 [13]	Military application
DS8	101	19572126	S17 [13]	Real time command and control system

4.1 Setup

We examine the predictive performance of the above total 4 approaches with 8 actual software fault count data in Table 2. The model parameters of those approaches are estimated by the maximum likelihood estimation. We use the first 20%, 50%, and 80% of the data points from the entire dataset for the training data, and the remaining data for the validation. We interpret these data points to represent the early, middle, and late software testing phases.

We apply the predictive mean absolute error (PMAE) as a meansure of predictive performance:

$$\text{PMAE} = \frac{1}{k}\sum_{i=1}^{k} |(n+i) - \tilde{\Lambda}(t_{n+i})|, \tag{12}$$

where $\{t_1, t_2, \ldots, t_n, \ldots, t_{n+k}\}$ represent the software fault time-detection data, k is the number

Algorithm 1 DAM (training)

Input:

Observed software time-domain data (training set) : $\boldsymbol{t} = \{t_1, t_2, \ldots, t_n\}$.

M candidate models : $\{\Lambda_i(t;\boldsymbol{\theta})\}, i = 1, 2, \ldots, M$.

Maximum training round P.

Initialization: Training round $p \leftarrow 1$, Resampling constant $\{r_i \leftarrow 1\}_{i=1,2,\ldots,n}$, Resampled data $\boldsymbol{t}_p = \{t_i^{(p)} \leftarrow t_i\}_{i=1,2,\ldots,n}$.

1: **repeat**

2: Select the minimum AIC model as a candidate prediction model $\tilde{\Lambda}_p(t;\tilde{\boldsymbol{\theta}}_p) \in \{\Lambda_i(t;\boldsymbol{\theta})\}$ among M candidate models with resampled data $\boldsymbol{t}_p$, where $\tilde{\boldsymbol{\theta}}_p$ is the maximum likelihood estimate with $\boldsymbol{t}_p$.

3: Calculate component of likelihood function in Eq.(3) CL_i:

$$CL_i = \lambda_p(t_i^{(p)};\tilde{\boldsymbol{\theta}}_p) \cdot e^{-\left[\Lambda_p(t_i^{(p)};\tilde{\boldsymbol{\theta}}_p) - \Lambda_p(t_{i-1}^{(p)};\tilde{\boldsymbol{\theta}}_p)\right]}, \tag{9}$$

where $\lambda_p(t_i;\tilde{\boldsymbol{\theta}}_p) = \frac{d\Lambda_p(t;\tilde{\boldsymbol{\theta}}_p)}{dt}$.

4: Calculate $MCL = \max\{CL_i\}, i = 1, 2, \ldots, n$.

5: Define likelihood loss functions $g(l_i, D)$:

$$g(l_i, D) = \frac{|MCL - CL_i|}{\max\{|MCL - CL_j|\}_{j=1,2,\ldots,n}}. \tag{10}$$

6: Calculate the loss $L_i = g(\Delta l_i, D)$ for sample $i = 1, 2, \ldots, n$ with any loss function $g(l_i, D)$.

7: Calculate an average loss : $\overline{L} = \sum_{i=1}^{n} L_i \rho_i$, where $\rho_i = r_i / \sum r_j$.

8: Calculate the confidence $\beta_p = \frac{\overline{L}}{1-\overline{L}}$.

9: Update the resampling constant $r_i \leftarrow r_i \cdot \beta_p^{(1-L_i)}, i = 1, 2, \ldots, n$.

10: Sort and relabel the resampling constants $\{r_i\}$ and associated samples $\{t_i\}$ as $\{r_{(i)}\}$ and $\{t_{(i)}\}$, such that

$$r_{(1)} < r_{(2)} < \ldots < r_{(n)}.$$

11: $p \leftarrow p + 1$.

12: The resampling training set $\boldsymbol{t}_p$ is built as

$$\{t_{(p)}, t_{(p+1)}, \ldots, t_{(n)}, t_{(n-p+1)}, t_{(n-p+2)}, \ldots, t_{(n)}\},$$

where the $p-1$ samples with the minimum resampling constant are removed and the $p-1$ samples with the maximum resampling constant are added.

13: **until** $\overline{L} \leq 0.5$ or $p > P$

Output:

p candidate prediction models $\{\tilde{\Lambda}_j(t)\}$ and associated confidence $\{\beta_j\}, j = 1, 2, \ldots, p$.

Algorithm 2 DAM (inference)

Input:

Observed software time-domain data : $\boldsymbol{t} = \{t_1, t_2, \ldots, t_n\}$.

Future testing time $t_u > t_n$.

P candidate prediction models $\{\tilde{\Lambda}_i(t)\}$ and associated confidence $\{\beta_i\}, i = 1, 2, \ldots, P$.

1: Obtain P prediction value $\Delta\tilde{y}_p = \tilde{\Lambda}_p(t_u) - \tilde{\Lambda}_p(t_n), p = 1, 2, \ldots, P$ for each candidate prediction model $\tilde{\Lambda}_p(t)$.

2: Each prediction value $\Delta\tilde{y}_p$ has an associated β_p.

3: Relabel $\{\Delta\tilde{y}_p\}$ while retain the associated $\{\beta_p\}$ as $\{\Delta\tilde{y}_{(p)}\}$ and $\{\beta_{(p)}\}$, such that

$$\Delta\tilde{y}_{(1)} < \Delta\tilde{y}_{(2)} < \ldots < \Delta\tilde{y}_{(P)}.$$

4: Assign confidence weight $c_{(p)} = \ln(\frac{1}{\beta_{(p)}}) / \sum \ln(\frac{1}{\beta_{(i)}})$ to the corresponding $\Delta\tilde{y}_{(p)}$.

5: Seek the finial interval prediction value $\tilde{y}_{(p^*)}$ where the indicator p^* is given by

$$p^* = \min\{\ p \mid \sum_{i=1}^{p} c_{(i)} \geq \frac{1}{2} \sum_{i=1}^{P} c_{(i)}\} \quad \text{(weighted median)}. \tag{11}$$

6: $\tilde{y} = \tilde{y}_{(p^*)} + n$.

Output: $\tilde{y}$, the expectation of cumulative number of software fault detected at testing time t_u.

of prediction terms, $\tilde{\Lambda}(t_{n+i})$ is the predicted value of cumulative number of software faults at t_{n+i} with any prediction approach. Comparing with other measures of predictive performance, *e.g.* PMSE, PMAE intuitively represents the sum of absolute errors between predicted values and actual data.

4.2 Predictive Performance

Table 3 $\sim$ Table 5 present the predictive performance (PMAE) results in Subsection 2.3 and Section 3, where *Minimum PMAE Model* representatives the model with the best predictive performance among 11 kinds of NHPP-based SRMs in Table 1, where we can not know the *Minimum PMAE Model* in advance. AMCM($\cdot$) denotes the AMCM with associated ($\cdot$) loss function in [10].

The results with yellow color in Table 4 $\sim$ 5 denote that AIC-based model section technique selected the best predictive performance among 11 candidate single models. It is seen that the minimum AIC technique could not select the best predictive performance in all of dataset at early software testing phase. Also, AIC-based model section technique could not always lead to the best predictive model in advance, especially in the early software testing phase.

The results with red color in Table 3 $\sim$ 5 present that the corresponding model has better

Table 3. Predictive performance of the existing approaches and our method at early software testing phase.

Data Set	DS1	DS2	DS3	DS4
Minimum PMAE Model	4.31 (LogEVMax)	8.65 (Pareto)	3.62 (LogLogist)	3.95 (Gamma)
Minimum AIC Model	14.02 (Exp)	13.08 (Exp)	7.66 (Gamma)	12.89 (Exp)
LWCM	18.45	12.08	22.92	7.90
AMCM (Linear)	20.98	**7.49**	22.48	22.81
AMCM (Square)	19.96	15.78	44.69	8.03
AMCM (Exp)	32.03	29.49	107.35	13.63
DAM	10.00	13.04	30.48	**3.08**
Data Set	DS5	DS6	DS7	DS8
Minimum PMAE Model	8.85 (LogNormal)	2.27 (Pareto)	3.77 (LogNormal)	16.03 (LogEVMax)
Minimum AIC Model	29.24 (TruncLogist)	8.51 (LogEVMax)	12.38 (Exp)	108.35 (Exp)
LWCM	25.09	10.95	40.59	72.48
AMCM (Linear)	12.59	11.20	6.47	25.26
AMCM (Square)	11.98	6.04	12.68	18.57
AMCM (Exp)	38.37	7.38	6.22	114.47
DAM	23.47	2.45	12.38	108.35

perdictive performance than baseline model with minimum AIC model. The results with bold mean that the corresponding model could outperform the minimum PMAE model, which was the single model with the best predictive performance. In early software testing phase, LWCM, AMCM(Linear) and AMCM(Square) improved the predictive performances in 4 of 8 cases and AMCM(Exp) gave the worst predictive results which could only improve the predictive performances in 2 of 8 cases. None of the existing models could improve the predictive performance in the majority of cases in early software testing phase. The similar results also could be seen in the middle and late software testing phases. The existing ensemble learning-based methods could not improve the predictive performances in most cases of all software testing phase. However, in early and middle software testing phases, our DAM improved the predictive performances, compared to the baseline model in 6 of 8 cases. Compared to the existing methods, our ensemble learning-based method could significantly improve the predictive performances.

5 Conclusion

In this paper, we concerned on the ensemble learning-based methods for software reliability prediction. We proposed an ensemble learning-based model, which was called the determinis-

Table 4. Predictive performance of the existing approaches and our method at middle software testing phase.

Data Set	DS1	DS2	DS3	DS4
Minimum PMAE Model	1.65 (LogNormal)	2.78 (Exp)	3.15 (Pareto)	3.93 (LogEVMax)
Minimum AIC Model	8.01 (Exp)	2.78 (Exp)	18.0 (Gamma) (Gamma)	6.80 (Exp) (Exp)
LWCM	7.56	5.97	24.37	71.18
AMCM (Linear)	11.76	88.79	21.64	56.52
AMCM (Square)	22.22	54.98	26.18	4.32
AMCM (Exp)	65.37	11.44	99.89	84.71
DAM	4.80	**2.28**	11.90	19.64
Data Set	DS5	DS6	DS7	DS8
Minimum PMAE Model	14.44 (LogEVMax)	3.01 (LogEVMax)	41.78 (Pareto)	24.65 (TruncEVMin)
Minimum AIC Model	17.14 (Exp)	3.01 (LogEVMax)	131.85 (TruncEVMin)	207.70 (Exp)
LWCM	**14.28**	5.60	135.60	54.87
AMCM (Linear)	15.98	9.81	41.79	96.54
AMCM (Square)	**9.16**	9.98	80.69	36.18
AMCM (Exp)	22.07	74.70	**6.54**	49.30
DAM	16.49	**2.63**	73.99	212.15

tic adaboosting model (DAM). Although there were several existing ensemble learning-based methods in the literature, the improvement of predictive performance with single model has not been examined yet. On the other hand, our DAM could significantly improve the predictive performance, compared to the single model approach in early and middle software testing phases.

In future, we are going to examine our ensemble learning-based methods for open-source software projects and develop the software reliability prediction methods based on the other ensemble learning-based methods.

6 Acknowledgements

This work was supported by the Research Institute for Mathematical Sciences, an International Joint Usage/Research Center located in Kyoto University. This work was supported by JST, the establishment of university fellowships towards the creation of science technology innovation, Grant Number JPMJFS2129.

Table 5. Predictive performance of the existing approaches and our method at late software testing phase.

Data Set	DS1	DS2	DS3	DS4
Minimum PMAE Model	2.38 (LogEVMax)	0.61 (LogEVMax)	2.75 (LogEVMax)	1.60 (Exp)
Minimum AIC Model	2.38 (LogEVMax)	0.61 (LogEVMax)	2.75 (LogEVMax)	1.60 (Exp)
LWCM	2.67	1.65	8.81	3.49
AMCM (Linear)	11.61	0.84	**2.22**	103.64
AMCM (Square)	2.43	1.16	**2.22**	234.12
AMCM (Exp)	**2.29**	6.73	21.20	37.88
DAM	2.38	0.66	3.92	2.11
Data Set	DS5	DS6	DS7	DS8
Minimum PMAE Model	2.94 (LogEVMax)	1.15 (LogEVMax)	2.28 (LogEVMax)	6.05 (LogEVMax)
Minimum AIC Model	4.53 (Gamma)	1.15 (LogEVMax)	2.63 (LogNormal)	9.93 (Pareto)
LWCM	**2.19**	2.47	3.06	9.26
AMCM (Linear)	20.60	1.77	2.55	9.78
AMCM (Square)	28.92	2.03	**1.45**	9.83
AMCM (Exp)	16.49	7.68	4.93	9.72
DAM	6.32	1.27	2.63	9.78

7 References

[1] A. A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood. Evaluation of competing software reliability predictions. *IEEE Transactions on Software Engineering*, SE-12(9):950–967, 1986.

[2] J. A. Achcar, D. K. Dey, and M. Niverthi. A bayesian approach using nonhomogeneous poisson processes for software reliability models. *Frontiers in Reliability*, pages 1–18, 1998.

[3] H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected Papers of Hirotugu Akaike*, pages 199–213. Springer, 1998.

[4] H. Bozdogan. Model selection and akaike's information criterion (aic): The general theory and its analytical extensions. *Psychometrika*, 52(3):345–370, 1987.

[5] H. Drucker. Improving regressors using boosting techniques. *Proceedings of the Fourteenth International Conference on Machine Learning (ICML)*, 97(107):e115, 1997.

[6] D. Fletcher. *Model Averaging*. Springer, 2018.

[7] Y. Freund and R. E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory. EuroCOLT 1995. Lecture Notes in Computer Science*, volume 904, pages 23–37, 1995.

[8] A. L. Goel and K. Okumoto. Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, R-28(3):206–211, 1979.

[9] S. S. Gokhale and K. S. Trivedi. Log-logistic software reliability growth model. *Proceedings of the Third IEEE International High-Assurance Systems Engineering Symposium (HASE-1998)*, pages 34–41, 1998.

[10] H. Li, M. Zeng, M. Lu, X. Hu, and Z. Li. Adaboosting - based dynamic weighted combination of software reliability growth models. *Quality and Reliability Engineering International*, 28(1):67–84, 2012.

[11] M. R. Lyu, editor. *Handbook of Software Reliability Engineering*. IEEE Computer Society Press and McGraw-Hill Book Company, 1996.

[12] M. R. Lyu and A. Nikora. A heuristic approach for software reliability prediction: the equally-weighted linear combination model. In *Proceedings. 1991 International Symposium on Software Reliability Engineering*, pages 172–181, 1991.

[13] J. D. Musa. Software reliability data. *Technical Report in Rome Air Development Center*, 1979.

[14] M. Ohba. Inflection s-shaped software reliability growth model. *Stochastic Models in Reliability Theory*, pages 144–162, 1984.

[15] K. Ohishi, H. Okamura, and T. Dohi. Gompertz software reliability model: Estimation algorithm and empirical validation. *Journal of Systems and Software*, 82(3):535–543, 2009.

[16] H. Okamura and T. Dohi. Srats: Software reliability assessment tool on spreadsheet. In *Proceedings of the 24th International Symposium on Software Reliability Engineering (ISSRE-2013)*, pages 100–107, 2013.

[17] H. Okamura, T. Dohi, and S. Osaki. Software reliability growth models with normal failure time distributions. *Reliability Engineering and System Safety*, 116:135–141, 2013.

[18] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[19] N. Sugiura. Further analysis of the data by akaike's information criterion and the finite corrections: Further analysis of the data by akaike's. *Communications in Statistics-theory and Methods*, 7(1):13–26, 1978.

[20] J. Wu, K. Daido, T. Dohi, and H. Okamura. Yet another least squares estimation in nhpp-based software reliability models with grouped data. In *IET Conference Proceedings CP886*, volume 2024, pages 205–211. IET, 2024.

[21] S. Yamada, M. Ohba, and S. Osaki. S-shaped reliability growth modeling for software error detection. *IEEE Transactions on Reliability*, R-32(5):475–484, 1983.

[22] M. Zhao and M. Xie. On maximum likelihood estimation for a general non-homogeneous poisson process. *Scandinavian Journal of Statistics*, 23(4):597–607, 1996.