Scientific Research Publishing

American Journal of Operations Research, 2022, 12, 111-125
https://www.scirp.org/journal/ajor
ISSN Online: 2160-8849
ISSN Print: 2160-8830

Reliability Assessment Tool Based on Deep Learning and Data Preprocessing for OSS

Shoichiro Miyamoto[1], Yoshinobu Tamura[1], Shigeru Yamada[2]

[1]Yamaguchi University, Yamaguchi, Japan
[2]Tottori University, Tottori, Japan
Email: c058vgw@yamaguchi-u.ac.jp, tamuray@yamaguchi-u.ac.jp, yamada@tottori-u.ac.jp

How to cite this paper: Miyamoto, S., Tamura, Y. and Yamada, S. (2022) Reliability Assessment Tool Based on Deep Learning and Data Preprocessing for OSS. *American Journal of Operations Research*, **12**, 111-125.
https://doi.org/10.4236/ajor.2022.123007

Received: April 10, 2022
Accepted: May 21, 2022
Published: May 24, 2022

Abstract

Recently, many open source software (OSS) developed by various OSS projects. Also, the reliability assessment methods of OSS have been proposed by several researchers. Many methods for software reliability assessment have been proposed by software reliability growth models. Moreover, our research group has been proposed the method of reliability assessment for the OSS. Many OSS use bug tracking system (BTS) to manage software faults after it released. It keeps a detailed record of the environment in terms of the faults. There are several methods of reliability assessment based on deep learning for OSS fault data in the past. On the other hand, the data registered in BTS differences depending on OSS projects. Also, some projects have the specific collection data. The BTS has the specific collection data for each project. We focus on the recorded data. Moreover, we investigate the difference between the general data and the specific one for the estimation of OSS reliability. As a result, we show that the reliability estimation results by using specific data are better than the method using general data. Then, we show the characteristics between the specified data and general one in this paper. We also develop the GUI-based software to perform these reliability analyses so that even those who are not familiar with deep learning implementations can perform reliability analyses of OSS.

Keywords

Open Source Software, Deep Learning, Software Reliability, Deep Learning, Software Tool

1. Introduction

Open source software (OSS) systems have been developed by various OSS project. Along with this, the number of users become increasing regardless of

individuals or companies. On OSS, the number of reported faults is also increasing. In particular, several serious faults are detected in the easier phase of operation. In addition, since the source code of OSS is opened to the public, it is easier for crackers to find vulnerabilities than commercial software. In particular, the zero-day attacks are targeted to vulnerabilities. The vulnerability will be fixed when the serious problems are recognized by users. This is one of the major risks by operating OSS on the internet.

Under these circumstances, it is necessary to assess the reliability of the OSS in order to operate OSS safely and stably. Many methods for software reliability assessment have been proposed by software reliability growth models [1] [2] [3]. However, it is difficult to assess the reliability of OSS. OSS user is increasing year by year. However, these users are not familiar with OSS. Therefore, the users may be exposed by the potential security and operational risks, if the OSS is in an unstable state. Various researchers have discussed the reliability of OSS to solve this problem [4] [5]. Software reliability model is one of the methods to measure OSS reliability. This method has the advantage for the reliability evaluation dynamically. Historically, the methods of software reliability assessment based on the software reliability growth models have been proposed by many researchers [1] [2] [3]. Also there are several methods of reliability assessment based on deep learning for OSS fault data in the past [6] [7].

On the other hand, a bug tracking system (BTS) is often used in OSS development to manage project progress by correcting faults. The BTS records many information in terms of the faults reported after releasing the OSS. There are the data such as the date, time, the nickname of reporter, assignee, OS, severity etc. on the BTS. This recorded data set is different according to OSS project. Also, there is the data collected in the specified OSS project only. Several researches have used BTS fault data to estimate the OSS reliability [6] [7]. However, there are no research papers in terms of the collected specific data to the OSS project.

We focus on specific data recorded on the BTS of OSS. We discuss the difference between the general data collected by BTS and the specified data. Furthermore, we develop GUI-based software based on deep learning by using the proposed method.

The organization of this paper is as follows:

Section 2: describes the several methods that have been proposed in previous papers.

Section 3: proposes the estimation method of OSS software reliability assessment based on deep learning. Then, the proposed method applies data recorded only in a specific OSS to deep learning.

Section 4: shows the optimizing method of neural network.

Section 5: describes the development of tools to perform deep learning.

Section 6: shows the several numerical examples based on proposed method and previous method.

Section 7: discusses the proposed method.

2. Data Preprocessing for OSS Fault Data and MTBF

There are various reliability models and neural networks for improving the software reliability. Many of the methods for estimating reliability with neural networks based on time-series analysis are using the past data of failure occurrence [8] [9]. In particular, many reliability estimation methods based on machine learning use only the software failure data in order to analyze the reliability. On the other hand, a reliability evaluation method using the BTS has been proposed in the past in terms of OSS software reliability. Also, the bug management by using BTS is widely used in OSS projects around the world. The general fault factors recorded in the BTS have been used in several researches [6] [7]. On the other hand, the BTS has collected the other specific data of OSS projects. This paper has the unique feature that we discuss the difference between the general data and the specific data on BTS. We show the general data recorded in many BTS in **Table 1**. **Table 2** shows the specific data of OSS. **Table 1** and **Table 2** are obtained from RedHad Openstack [10].

In order to use these data in deep learning, it is necessary to convert above mentioned data to numerical values. In this paper, the general data have been converted into numerical values as shown in **Table 3**. Similarly, the specific data have been converted into numerical values as shown in **Table 4**.

Mean Time Between software Failures (MTBF) is one of reliability assessment measures. The software is the reliable system if the MTBF becomes large. Then the software will operate continuously without failure. The MTBF grows as development progresses because the software faults are removed during the software operation. i-th MTBF_i is given by the following equation

Table 1. The list of general fault data in BTS.

Classification	Contents
Opened	The date and time recorded on the bug tracking system.
Changed	The modified date and time.
Status	The fixing status of fault.
Resolution	The status of resolution of fault.
Hardware	The name of hardware under fault occurrence.
OS	The name of operating system under fault occurrence.
Severity	The level of fault.
Summary	The brief contents of fault.
Reporter	The nickname of fault reporter.
Assignee	The nickname of fault assignee.
Product	The name of product included in OSS.
Component	The name of component included in OSS.
Version	The software version number of OSS.

Table 2. The list of specific fault data in BTS.

Classification	Contents
Votes	The number of faults in BTS.
Type	The type of faults.
Keywords	The keyword about fault.
QA Contact	The name of correspondent person of fault.
QA Contact Real Name	The real name of correspondent person of fault.
Alias	The name of fault alias.
Dependent Products	The product depending to fault.
Depends On	The faults depend on the other fault.
Docs Contact	The nickname of fault reporter.
Fixed In Version	The version with fixed fault.
URL	The URL related to fault.

Table 3. Method of numeric conversion for each factor in general data.

Factor	Method of Numeric Conversion
Opened, Changed	Time of between previous and current fault
Product, Component, Version, Reporter, Assignee, Severity, Status, Resolution, Hardware, OS	Frequency encoding
Summary	Count encoding

Table 4. Method of numeric conversion for each factor in specific data.

Factor	Method of numeric conversion
Opened, Changed	Time of between previous and current fault
votes, Type, Keywords, Classification QA Contact, QA Contact Real Name, Alias, Dependent Products, Depends On, URL	Frequency encoding
Fixed In Version, Docs Contact	Count encoding

$$\mathrm{MTBF}_i \equiv \frac{T_i}{N_i}, \tag{1}$$

where each parameters are defined as follows:

N_i: The total number until i-th fault occurred during operation,

T_i: The time of software operation until i-th fault.

We consider that MTBF is used for the systems that can be repaired. On the other hand, the Mean Time To Failure (MTTF) is used for the non-repairable systems. In this paper, MTBF is used as the assessment measure for OSS reliabil-

ity because the OSS is repairable system. In this paper, the MTBF is defined as the time between the correction of a fault and the occurrence of the next fault.

3. Estimation of OSS Reliability Based on MTBF and Deep Learning

Several researchers have proposed the useful deep learning algorithms. In this paper, we use MTBF-based deep learning to estimate OSS reliability. In this paper, we have constructed a feed-forward neural network in which signals propagate in the order of input layer, intermediate layer, and output layer, respectively. Then, we distinguish between the general data recorded on BTS and the specific data on OSS project. We show three cases based on deep learning in **Table 1** and **Table 2** as explanatory variables. Moreover, we consider the case of the case with both general and special data as explanatory variables.

4. Optimization of Deep Neural Network

Several algorithms have been proposed by several researchers for parameter optimization of deep learning networks [11] [12] [13] [14] [15]. In this paper, we use the Adam optimizer well known as deep learning optimization algorithm [16]. **Figure 1** shows the details of the hyper-parameters used for Adam. Adam is a stochastic gradient descent method that improves on AdaGrad and RMSProp. It has the characteristic of updating values more frequently where the gradient of the loss function is large and reducing the update range. Then, the gradient is highly variable, the facilitating convergence with a small amount of computation.

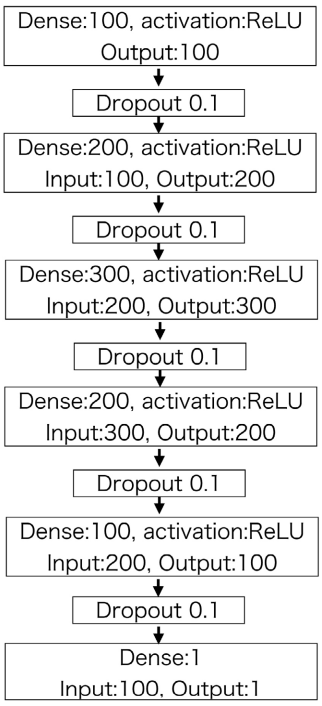

Figure 1. Network model.

4.1. Activation Function

Activation function is the function used to convert the sum of inputs from the network into outputs. This function is typically a nonlinear function in order to achieve in Figure 1 outputs that cannot be represented by a single node. In this paper, ReLU is used as the activation function. ReLU is following as

$$f(u)=\begin{cases}0 & u\le 0,\\ u & u>0.\end{cases} \tag{2}$$

In deep learning, the gradient method defines a loss function as Equation (2). Also, there is the other loss function with weights to modify the derivative toward near 0. This is often proposed when learning proceeds. However, depending on the activation function, as the number of layers increases, nodes with gradients close to 0 are more likely to appear, and the Vanishing gradient problem, where the weights are not updated, is more likely to occur. ReLU is a function that is less prone to this problem. In mathematics, a value of 0 is not differentiable. In general, when the value is 0, including when it is less than 0, it follows that

$$f'(u)=\begin{cases}0 & u\le 0,\\ 1 & u>0.\end{cases} \tag{3}$$

From Equation (3), the gradient is sufficiently large to prevent the gradient loss problem. It also has the advantage of being less computationally expensive due to its simple formulas.

4.2. Loss Function

Loss function is a function that evaluates the error between the output value of the network and the correct value when optimizing the weights of each node in deep learning. In this paper, the estimation is a regression problem to predict the output value of MTBF. The widely known loss functions for regression problems are MSE (Mean Squared Error), which squares the error between the correct solution and the output value, and MAE (Mean Absolute Error), which takes the absolute value of the correct solution and the output value, respectively. MSE uses the error squared, making MSE a more error-tolerant network than MAE. Since there are many outliers in the MTBF estimated in this paper that are far from the overall trend, MAE is used as the loss function. MAE is following as

$$\mathrm{MAE}=\frac{1}{N}\sum_{i=1}^{N}\left|\widehat{y_i}-y_i\right|, \tag{4}$$

where each parameter is as follows:

N: the total number of data,

$\widehat{y_i}$: estimate,

y_i : output data from network.

5. Development of Prototype Deep Learning Tool Based on GUI

Various research papers have used the deep learning to estimate the reliability of

OSS. In order to perform such deep learning, there are time constraints for users who are not familiar with programming language, such as the need to build a scientific environment for program installation, and computers for deep learning, as well as time constraints for building the environment. This makes it difficult for the average user to understand the reliability of OSS by using deep learning. Therefore, we have developed a GUI-based reliability analysis tool to solve this problem. Our software works as follows

1) The data of Input OSS fault data are send to Node.js server from client software.

2) The Python read the transfer data, and start the algorithm of deep learning.

3) The Python send the estimation results to Node.js server.

4) The Python transfer the estimation results to client software. Then, the user can see the results by several graphs.

The software workflow is shown in Figure 2. The environment for deep learning is not built on the user's side, but on the server, making the software independent of the user's execution environment. The program operation screen is shown in Figure 3. The intuitive GUI-based software can be used by users who are not familiar with programming. The method proposed in Section 3 is used for deep learning. In addition, as shown in Figure 4, the user can choose whether to use general data, specific data, or both as parameters for deep learning. This enables optimal deep learning at the user's discretion. After the training is completed, the server returns three graphs, MTBF, cumulative MTBF, and the Error Scatter, respectively. Then, the user can view the estimation results using the application. This paper presents as the examples for numerical results of the method proposed in Section 3 using the prototype tool.

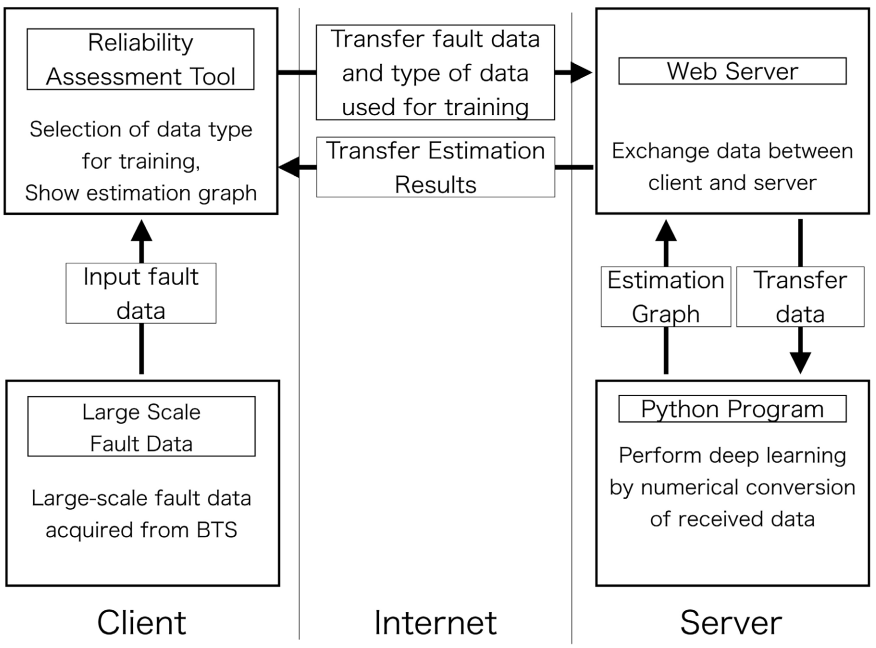

Figure 2. The workflow of reliability assessment tool.

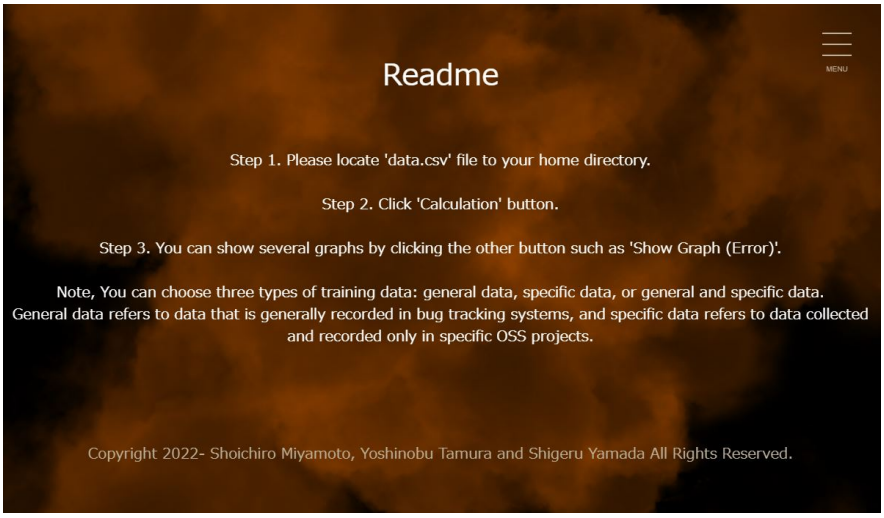

Figure 3. The screen of software readme.

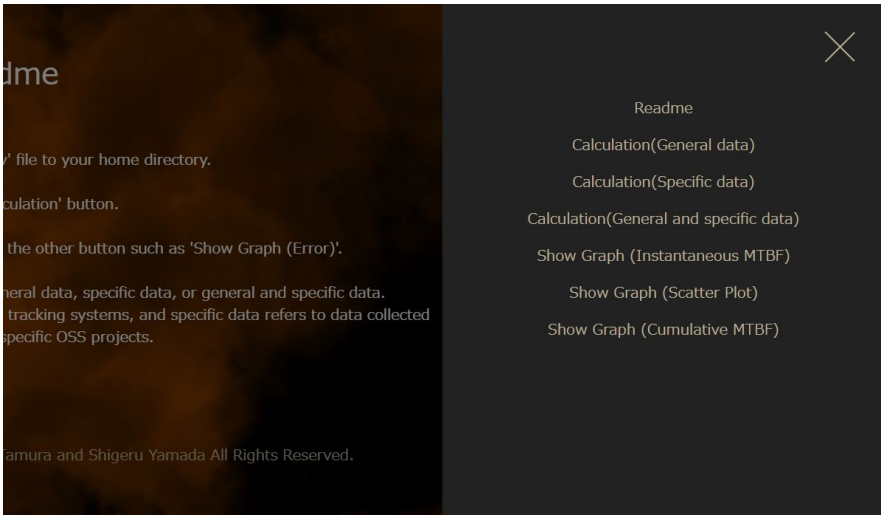

Figure 4. The screen of software menu.

6. Numerical Illustrations of Our Tool

We have retrieved 20,000 large fault data sets from RedHat Openstack's BTS and input them into our deep learning tool. Then, the results were obtained from three deep training runs: one with general data, one with specific data, and one with both general and specific data, respectively. **Figures 5-7** show the estimated instantaneous MTBF. **Figure 5** and **Figure 7** show that the estimation results generally capture the trend of the testing data. On the other hand, the graph in **Figure 6** shows that the estimation result of MTBF is not so large even though the MTBF of the testing data is higher.

Figures 8-10 show the estimated cumulative MTBF. The cumulative MTBF shows the MTBF when all the instantaneous MTBFs up to a certain point are summed. The graphs in **Figure 8** and **Figure 10** show that the testing data is well estimated for the actual data. In particular, the shape of the graphs of the testing data and the estimation results in **Figure 10** are very similar. On the other hand,

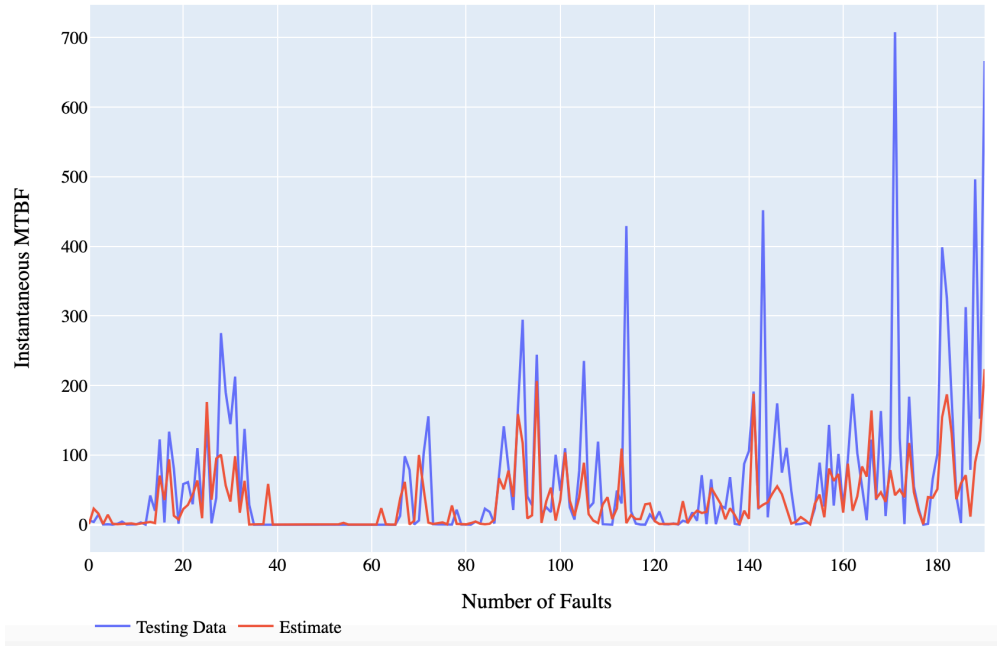

Figure 5. The estimated MTBF for faults using general parameter.

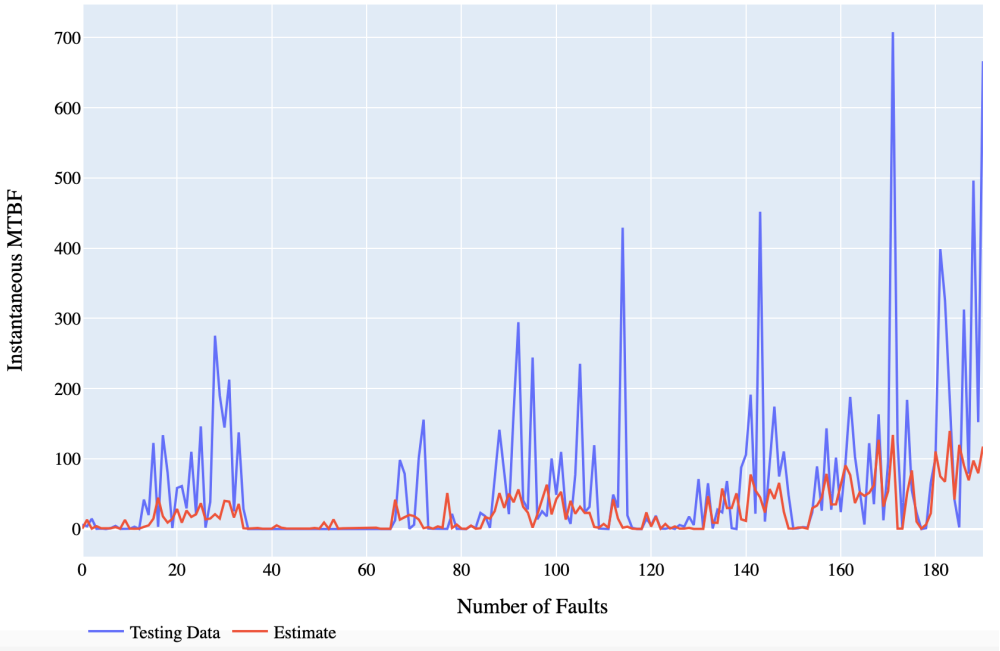

Figure 6. The estimated MTBF for faults using specific parameter.

the graph in **Figure 9** shows a lower level of cumulative MTBF than the testing data.

Figures 11-13 show the scatter of errors. Also, **Figure 11** and **Figure 12** show that the testing data and estimation results tend to plot near the same when the MTBF of the testing data is low. We found that the estimate is not well done when the MTBF of the testing is high. **Figure 13** plots the testing data and the estimation results in the same vicinity regardless of the testing data.

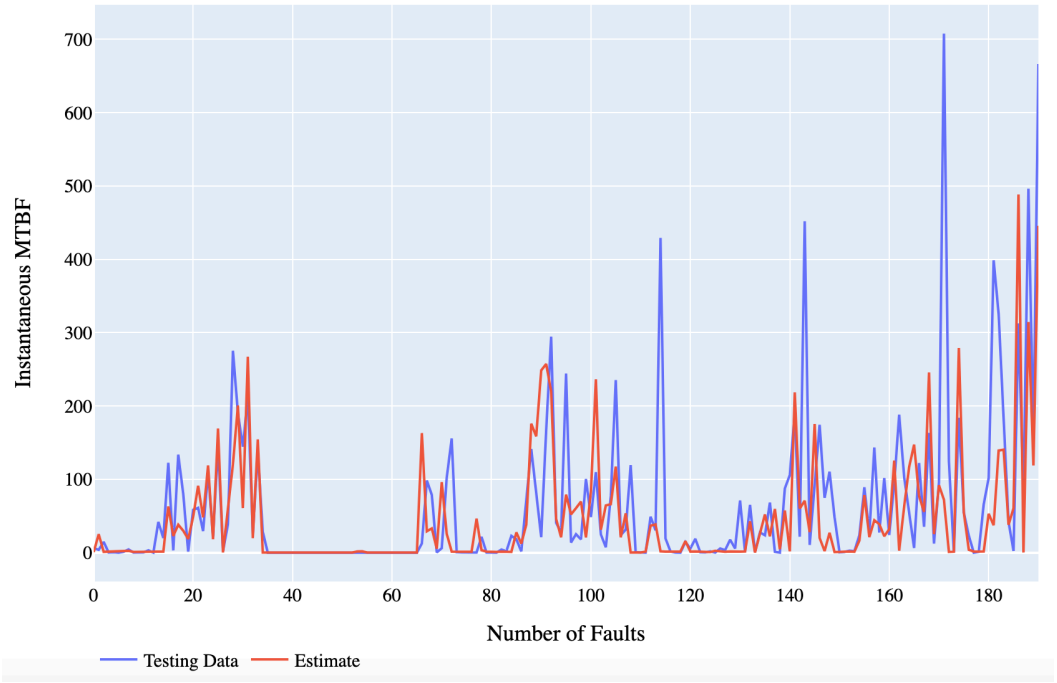

Figure 7. The estimated MTBF for faults using general and specific parameter.

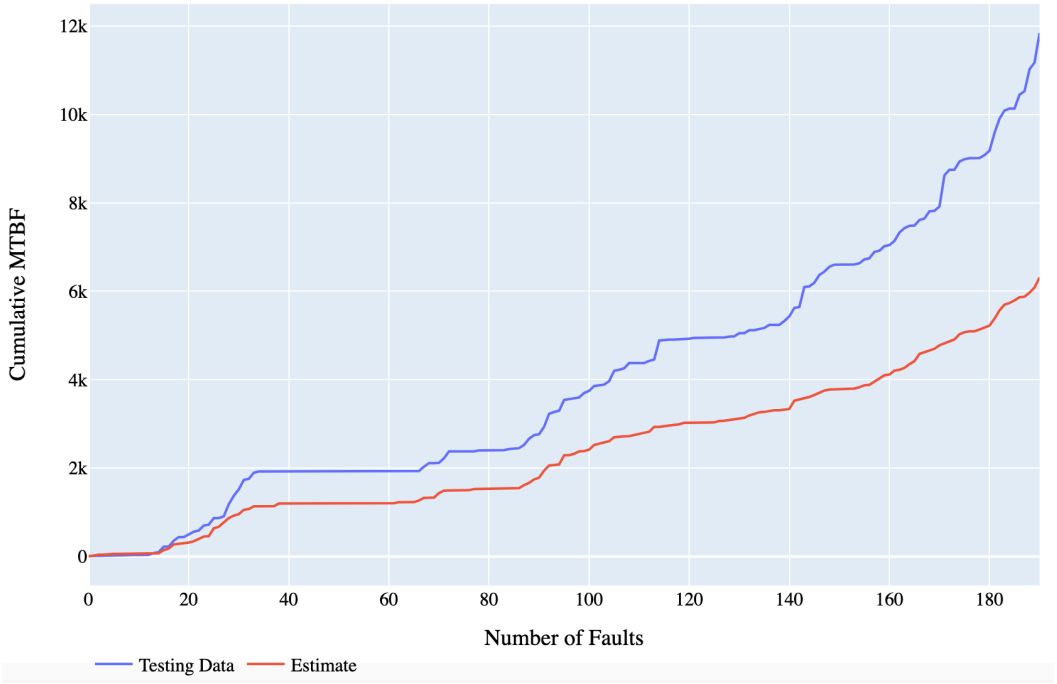

Figure 8. The estimated cumulative MTBF for faults using general parameter.

Table 5 shows the correlation coefficient for the scatter of errors. The correlation coefficients are the highest for the general and specific data. This indicates that the accuracy of estimation is the highest for the general and specific data.

From above results, we find that combining general and specific data for learning may provide more accurate estimation than using general data.

Table 5. The correlation coefficient of errors.

General data	0.60552
Specific data	0.58601
General data and specific data	0.61113

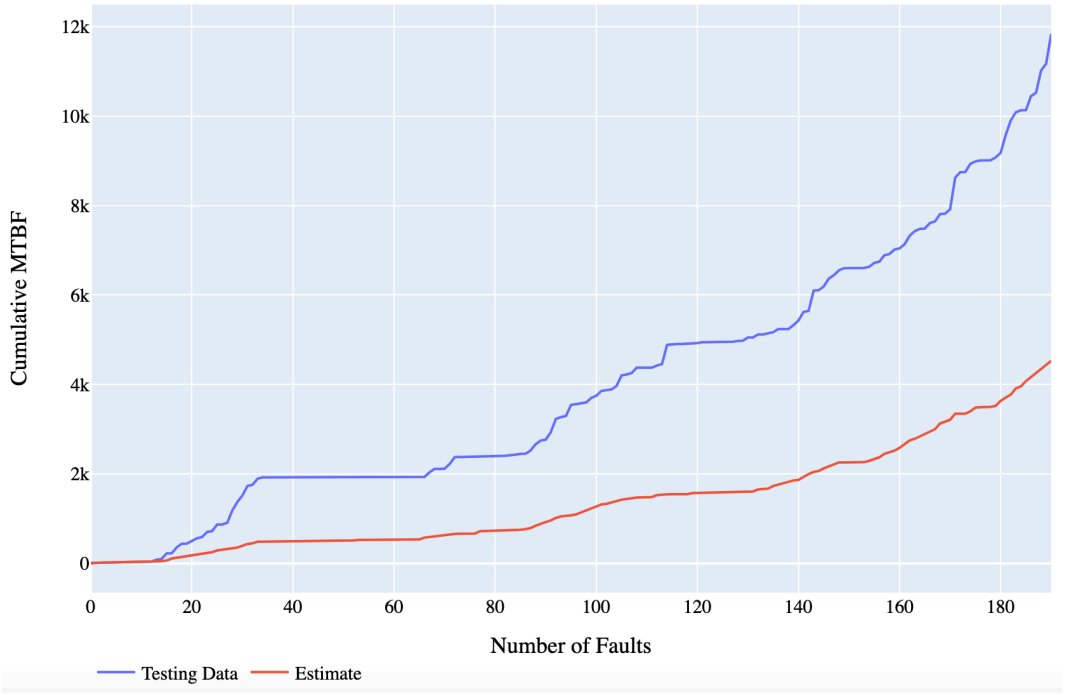

Figure 9. The estimated cumulative MTBF for faults using specific parameter.

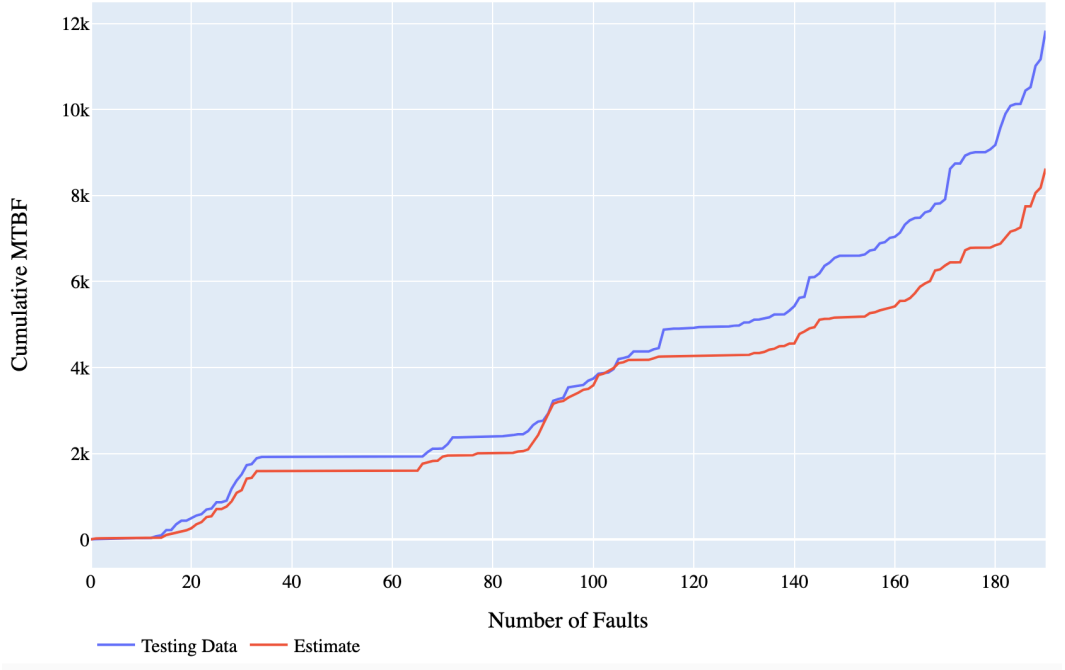

Figure 10. The estimated cumulative MTBF for faults using general and specific parameter.

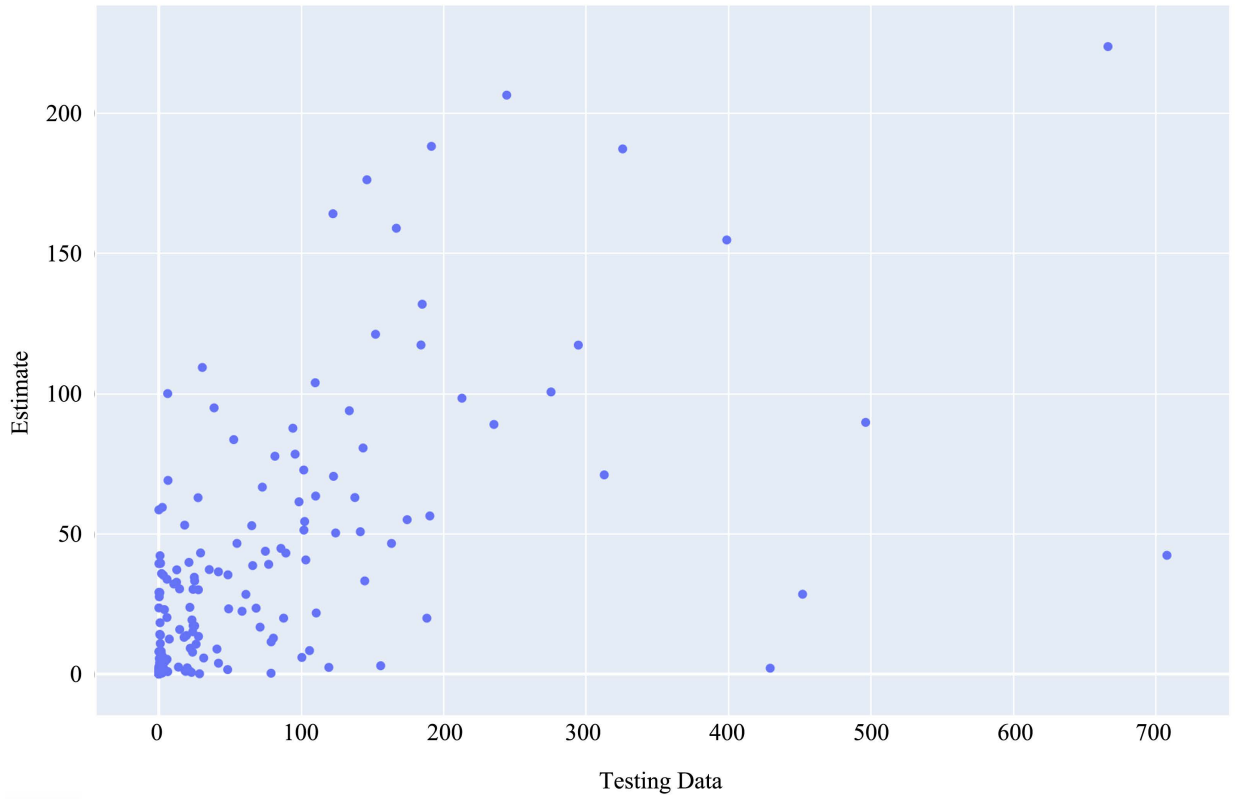

Figure 11. The relation of testing data and estimate for faults using general parameter.

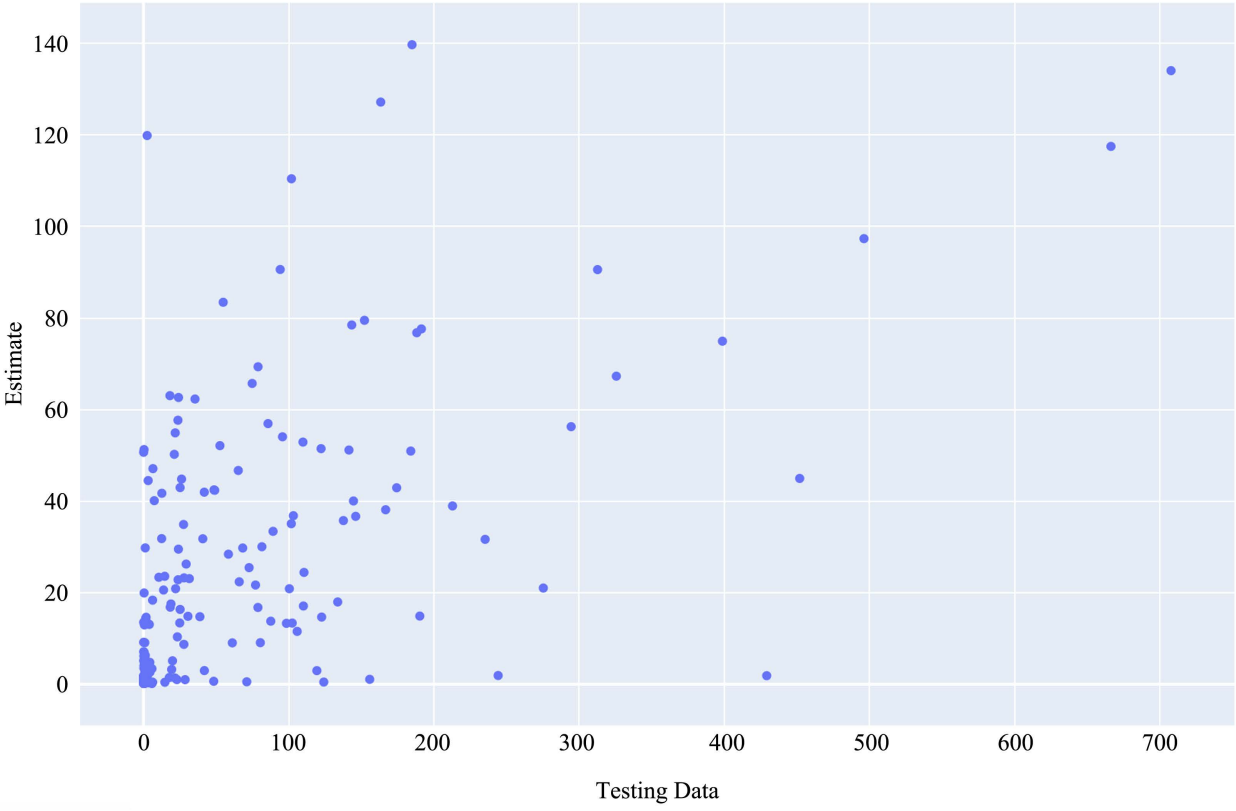

Figure 12. The relation of testing data and estimate for faults using specific parameter.

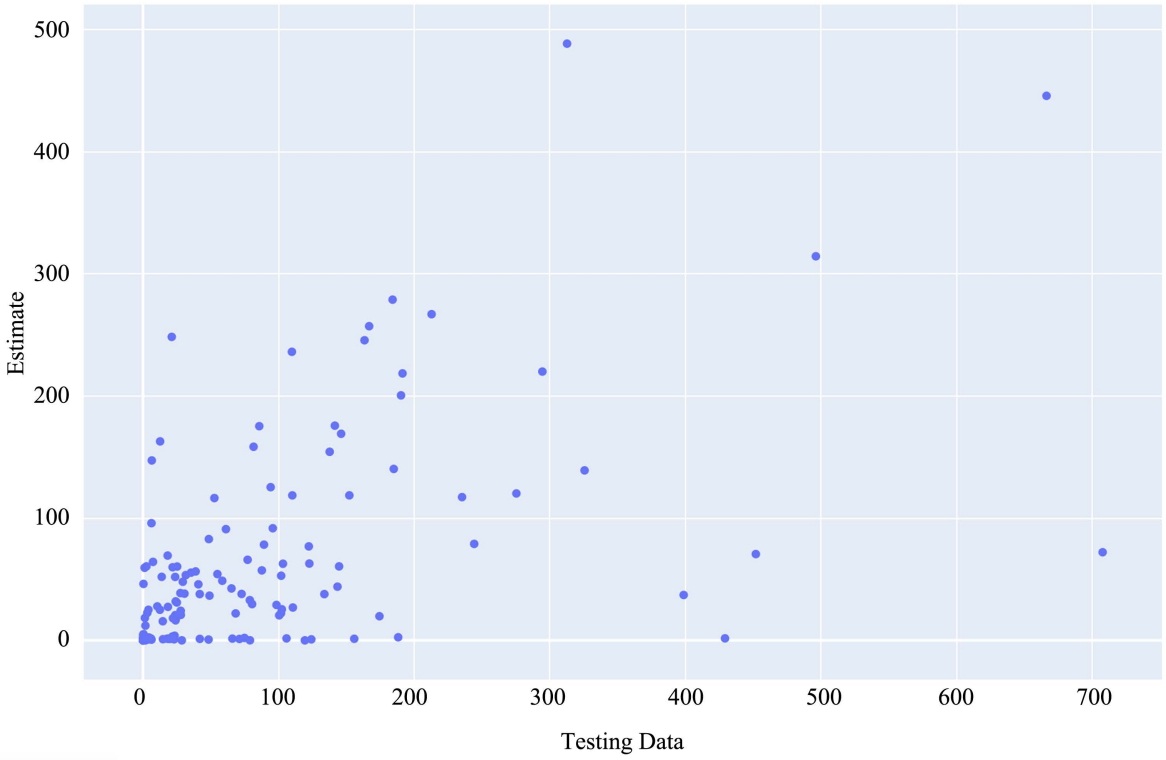

Figure 13. The relation of testing data and estimate for faults using general and specific parameter.

7. Concluding Remarks

In this paper, we have discussed the availability of our method by using specific data for estimate MTBF. As a result, the proposed estimation results using the specific data are not well than the general data. The estimation using a combination of general and specific data was more accurate than the estimation using general data. This has shown that specific data is useful as an explanatory variable with different characteristics from general data.

In addition, we have developed a reliability assessment tool based on deep learning that can be used by users who are not familiar with deep learning. The tool has been designed to be more practical by allowing users to choose whether to use the parameters recorded only in the specific OSS projects proposed in this paper.

In BTS, there are various types of failure data other than those used in this paper. Therefore, as a future study, we are planning to investigate the construction of reliability models by using fault data collected in BTSs of other OSS projects. On the other hand, some specific data collected from OSS projects may adversely affect the estimation. We would like to discuss the selection of appropriate specific data in the future.

Acknowledgements

This work was supported in part by the JSPS KAKENHI Grant No. 20K11799 in Japan.

Conflicts of Interest

The authors confirm that there is no conflict of interest to declare for this publication.

References

[1] Lyu, M.R. (Ed.) (1996) Handbook of Software Reliability Engineering. IEEE Computer Society Press, Los Alamitos, CA.

[2] Yamada, S. (2014) Software Reliability Modeling: Fundamentals and Applications. Springer-Verlag, Tokyo. https://doi.org/10.1007/978-4-431-54565-1_1

[3] Kapur, P.K., Pham, H., Gupta, A. and Jha, P.C. (2011) Software Reliability Assessment with OR Applications. Springer-Verlag, London. https://doi.org/10.1007/978-0-85729-204-9

[4] Han, X. (2021) A Study of Performance Testing in Configurable Software Systems. *Journal of Software Engineering and Applications*, **14**, 474-492. https://doi.org/10.4236/jsea.2021.149028

[5] Onarcan, M. and Fu, Y.J. (2018) A Case Study on Design Patterns and Software Defects in Open Source Software. *Journal of Software Engineering and Applications*, **11**, 249-273. https://doi.org/10.4236/jsea.2018.115016

[6] Tamura, Y. and Yamada, S. (2018) AI Approach to Fault Big Data Analysis and Reliability Assessment for Open-Source Software. In: Anand, A. and Ram, M., Eds., *System Reliability Management*: *Solutions and Technologies*, *Advanced Research in Reliability and System Assurance Engineering*, CRC Press, Boca Raton, 1-17. https://doi.org/10.1201/9781351117661-1

[7] Tamura, Y., Ueki, R., Anand, A. and Yamada, S. (2020) Estimation of Mean Time between Failures Based on Deep Feedforward Neural Network for OSS Fault Big Data. *Proceedings of the* 4*th International Conference on Mathematical Techniques in Engineering Applications*, Dehradun, 4-5 December 2020, 1-12.

[8] Karunanithi, N., Whitley, D. and Malaiya, Y.K. (1992) Using Neural Networks in Reliability Prediction. *IEEE Software Magazine*, **9**, 53-59. https://doi.org/10.1109/52.143107

[9] Dohi, T., Nishio, Y. and Osaki, S. (1999) Optimal Software Release Scheduling Based on Artificial Neural Networks. *Annals of Software Engineering*, **8**, 167-185. https://doi.org/10.1023/A:1018962910992

[10] The OpenStack Project, Build the Future of Open Infrastructure. https://www.openstack.org/

[11] Kingma, D.P., Rezende, D.J., Mohamed, S. and Welling, M. (2014) Semi-Supervised Learning with Deep Generative Models. *Proceedings of the* 27*th International Conference on Neural Information Processing Systems*, Montreal, 8-13 December 2014, 3581-3589.

[12] Blum, A., Lafferty, J., Rwebangira, M.R. and Reddy, R. (2004) Semi-Supervised Learning Using Randomized Mincuts. *Proceedings of the* 21*st International Conference on Machine Learning*, Banff, 4-8 July 2004, 13-20. https://doi.org/10.1145/1015330.1015429

[13] George, E.D., Dong, Y., Li, D. and Alex, A. (2012) Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition. *IEEE Transactions on Audio*, *Speech*, *and Language Processing*, **20**, 30-42. https://doi.org/10.1109/TASL.2011.2134090

[14] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.A. (2010) Stacked

Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research*, **11**, 3371-3408.

[15] Martinez, H.P., Bengio, Y. and Yannakakis, G.N. (2013) Learning Deep Physiological Models of Affect. *IEEE Computational Intelligence Magazine*, **8**, 20-33. https://doi.org/10.1109/MCI.2013.2247823

[16] Kingma, D.P. and Ba, J.L. (2015) Adam: A Method for Stochastic Optimizations. *Proceedings of the International Conference on Learning Representations*, San Diego, 7-9 May 2015, 1-15.