Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 1276830, 8 pages
https://doi.org/10.1155/2022/1276830

*Research Article*

# Defect Prediction Technology of Aerospace Software Based on Deep Neural Network and Process Measurement

**Tianwen Yao, Ben Zhang, Jun Peng, Zhiqiang Han, Zhaobing Yang, Zhi Zhang, and Bo Zhang**

*Systems Engineering Institute of Sichuan Aerospace, Chengdu 610100, China*

Correspondence should be addressed to Tianwen Yao; supermantw@126.com

Received 18 April 2021; Revised 25 September 2021; Accepted 28 December 2021; Published 1 February 2022

Academic Editor: Antonio M. Gonçalves de Lima



In order to ensure high reliability, the efficiency of traditional aerospace software testing is often low. With the rapid development of machine learning, its powerful data feature extraction ability has great potential in improving the efficiency of aerospace software testing. Therefore, this paper proposed a software defect prediction method based on deep neural network and process measurement. Based on the NASA data set and combined with the software process data, the software defect measurement set is constructed. 35 measurement elements are used as the original input, and multiple single-layer automatic coding networks are superimposed to form the deep neural network model of software defect. The model is finally trained by the layer-by-layer greedy training method to realize software defect prediction. Experimental verification shows that the prediction method has a good prediction effect on aerospace software defects, and the accuracy rate reached 90%, which can greatly improve the efficiency and effect of aerospace software testing.

## 1. Introduction

With the continuous development of aerospace technology, the scale and complexity of aerospace software are getting higher and higher, and the development cycle of aerospace models is gradually shortening. This poses new challenges to the development process of software in the aerospace field. However, in order to ensure high security and stability, aerospace software development has a strict development process, which covers the stages of requirement analysis, outline design, detailed design, coding, and testing. Among those processes, the testing is an important and indispensable stage to ensure the quality of software [1]. Software testing could be divided into static testing, unit testing, assembly testing, configuration item testing, and software system testing. These testing items are to ensure that the users' requirements for the software are met. Statistics show that the cost of discovering and repairing defects increases exponentially over time [2]. For example, it takes an average of 2 to 3 minutes to fix a software defect in the static testing phase, 10 to 20 minutes in the unit testing phase, 1 to 2 hours in the assembly testing phase, and 40 to 50 hours in the system testing phase. Therefore, how to dig out defective modules in the early stage of software development has become an urgent problem to be solved.

In order to realize early detection of software defects, software defect prediction technology has become an important research topic in software engineering. Since the 1970s, researchers have begun to build metric set based on the software history warehouse and used statistics and machine learning methods to study software defect prediction. These studies mainly focused on the following two aspects: (1) research on software defect data measurement set and its construction method; (2) research on construction technology of software defect prediction model.

For the first aspect, the realization of defect prediction requires a large amount of defect data to study the relationship between module metrics and module defects. Thus, it is important to obtain effective defect metrics. In early studies, most of them are based on the assumption that "the higher the code size and complexity of the software module, the more likely it is to have defects" to design metrics for

software modules and the inherent complexity of the program, for example, the line of code [3] (LOC) to measure the size of software, the Halstead [4], and McCabe [5] measurement method to measure the complexity of software, etc. Halstead is a method to measure the complexity of a program based on the total number of operators and operands in the program, and McCabe is based on the complexity of program topology.

For the second aspect, most researches on the construction of software defect prediction models are based on machine learning methods. As the performance of different machine learning methods is different, researchers have studied how to build software defect prediction models from different perspectives. Catal and Diri [6] used NASA data sets to study the influence of defect metric, defect data size, and feature set selection methods on the prediction performance of the model. Using AUC (Area Under Curve) as the evaluation metric, they concluded that, on large-scale data sets, the RF (random forest) method has the best performance; on a small-scale data set, the BN (Bayesian network) method has the best effect. Lessmann et al. [7] used the NASA data set to deeply compare 22 machine learning methods in 6 categories, mainly including nearest neighbor method, statistical method, decision tree, support vector machine, neural network, and ensemble learning method. Using AUC as evaluation metrics, the research has found that there is no significant performance difference between most of the best machine learning methods. In addition, some of the latest results in machine learning have also been applied to the construction of defect prediction models. For example, Lu et al. [8] used active learning to build a defect prediction model. When they trained the model, they were based not only on the training data of the previous software version, but also on selecting and marking a small number of examples from the current software version. It can be seen from current researches that when constructing a software defect prediction model, it is necessary to select an appropriate machine learning method according to the characteristics of the predicted object.

From the above analysis, it can be seen that the early software defect prediction is mainly based on statistical learning, through which the number of software defects can be obtained and the number of possible defects can be predicted statistically. After the 1990s, researchers began to apply machine learning to software defect prediction, which made the performance of software defect prediction greatly improved. In recent years, a large number of studies have proved the applicability of machine learning for software defect prediction [2, 9, 10]. Although machine learning-based software defect prediction technologies are becoming more and more intelligent, they are always exploring the relationship between software metrics and software defects. In the aerospace field, the software has extremely high requirements for safety and reliability. Efficient software testing is a key step to ensure the quality of aerospace software. However, even if some problems can be found through testing, it is difficult to find some subtle and implicit software errors. Therefore, how to efficiently and reliably carry out software testing and effectively dig out software defects has become an inevitable problem. Pfleeger [11] pointed out that the quality model is based on the information contained in the software metrics; thus choosing a set of standardized software metrics matters much in constructing the software quality model. There is also literature [12] dedicated to reducing different types of noise in software metrics, so as to select the optimal metric set, thereby improving the prediction performance.

In order to improve the efficiency of aerospace software defect model, this paper analyzed the characteristics of aerospace software and used machine learning methods to study the defect prediction of aerospace software and proposed a software defect prediction method based on deep neural networks and process metrics. Compared with other methods of the latest research, the main advantages of this study are as fallows. (1) When constructing the software defect measurement model, the process data in the aerospace software development is introduced into the data set to form a set of software defect metrics. (2) Set up an autoencoder according to the metrics, and establish a deep neural network model. (3) Carry out defect prediction for the current software version by training the test data of the historical software version to achieve high defect prediction accuracy. The main research in the content includes designing a set of software defect metrics for the aerospace field and constructing a deep neural network model for aerospace software defect prediction.

## 2. Metric Set for Aerospace Software

*2.1. The Design of Software Defect Metrics.* In the process of software defect prediction, the three main factors that affect predictive performance are designing defect metrics, processing data set, and building predictive models [6]. Among them, design metrics are the basis of software defect prediction [13].

It can be seen from the discussion in the last section that, in the process of software development, process factors are closely related to software defects, such as the developer's experience, the organizational structure of the project, source code change characteristics, and dependencies between software modules. In the aerospace software defect prediction, how to mine these process factors to design effective metrics is the main problem. To handle this problem, this paper designs a "product + process" defect metric set for aerospace software, which is based on the metrics used in NASA software data set and combined with the characteristics of aerospace software.

*2.2. NASA Software Data Metrics.* Obtaining effective defect data set is a prerequisite for the construction of software defect prediction model. Defect data warehouse collects defect data of software in the life cycle stages of requirements, design, development, testing, etc. Therefore, choosing a suitable defect data warehouse would affect the performance of the prediction model. The NASA defect data warehouse (NASA ddw) is a set of software defect data disclosed by NASA [14]. The project types include ground

control software system, flight control software system, and flight monitoring software system. NASA ddw is a public defect data warehouse, in which both meta and error information can be used, Thus, it is very suitable for the training and prediction model construction of aerospace software defects.

The NASA data set contains 21 software metrics, which are composed of the classic metrics of Halstead and McCabe plus the number of lines of code (LOC). Table 1 shows the specific metric symbols and definitions.

*2.3. Process Metrics for Aerospace Software.* Process measurement is a measurement of various factors that are closely related to the software development process. The defect metric design of the software development process provides a quantitative measurement for the evaluation and control of the software development process. Compared with other software, aerospace software has strong field characteristics due to its special application. Both the development and testing of aerospace software are based on the following characteristics:

(1) Aerospace software complies with a fairly uniform development model; besides, its design reference and development standards are unified, with detailed process records in all planning, development, testing, maintenance, and change processes.

(2) In the development phase of aerospace software, only a small amount of mission requirement changes require software version upgrades, so that each software configuration item will form multiple versions during the development cycle.

(3) The personnel of the aerospace software development team usually changes little, which makes the inheritance between software versions higher.

(4) The development cycle of aerospace missions is usually long, in which a large number of tests could be carried out, so the software could accumulate a large amount of data for research and analysis during these experiments.

Some studies have pointed out that the historical changes of software are related to the occurrence of software defects. For example, Wahyudin et al. [15] confirmed that the tendency of software defects is connected with the number of software changes and the number of code lines. Considering the above characteristics, this paper uses the number of changes, the proportion of new developers in the change, and the total number of lines of code added, deleted, or modified as the process metrics of the historical changes of aerospace software. At the same time, in terms of software management, there is a certain correlation between software defects and management processes, such as software requirements and personnel ratio management. As shown in Table 2, this paper proposes 10 management process metrics for aerospace software, which covers the management process of requirements analysis phase, design phase, and coding phase. Therefore, Tables 1 and 2 are the software metric set used in this paper.

## 3. Construction Strategy of Aerospace Software Defect Prediction Model

With the development of software technology, the acquisition and storage of high-dimensional data become easier, and the information data is continuing to expand and become larger. On the one hand, it symbolizes that people are becoming more and more aware of data, and data analysis is becoming more and more comprehensive; on the other hand, the processing of high-dimensional data has also become a major challenge for the industry; that is, when researchers use machine learning technology for data research, it usually faces the Curse of Dimensionality [16]. The Curse of Dimensionality was first proposed by Bellman when considering optimization problems. It is used to describe the data in the high-dimensional space which encounters various problem scenarios due to the increase of the volume index. Its manifestation in machine learning is that the demand for the number of training samples increases exponentially with the dimensionality of the feature space. Therefore, in software defect prediction, there is usually a situation that the number of training samples is insufficient due to the high dimension of the feature space, which in turn leads to the phenomenon of "overfitting" [2].

*3.1. Data Dimension Reduction by Autoencoder.* Autoencoder [17] is an unsupervised learning algorithm. Its goal is to learn from the input data to optimize the feature value of these data, while using backpropagation algorithm to make the target value as close to the input value as possible. Through continuous training to adjust the parameters in the network, the training weight value of each layer in the network can be obtained. An autoencoder is a three-layer neural network, that is, an input layer, a hidden layer, and an output layer. The number of neurons in the input layer is the same as that of the output layer, and the number of middle layers is smaller than the input layer. The middle layer is used as a compressed representation of the input layer, that is, the feature after we reduce the dimensionality of the input data. As shown in Figure 1, for a single self-encoding network, the feature reduction of input data is mainly divided into two stages: encoding and decoding.

When encoding, let each input sample be represented by a vector, as follows:

$$(x_1, x_2, \ldots, x_n). \tag{1}$$

In the formula, $x_1 \sim x_n$ are the values of the different metric in each sample and $n$ is the number of aerospace software metrics. There are metrics 36 in this paper.

The final output of the network is a probability value in the range of 0 to 1. However, because the numerical unit of each metric element is different, it is necessary to scale the nonproportional metric value of the data set. The specific calculation method is to divide the specific value of the metric by the maximum value of the metric value to obtain the numerical ratio, whose value range is [0,1], which meets the output requirements.

Table 1: The metrics of NASA data set.

| Metric | Definition | Metric | Definition |
|---|---|---|---|
| loc | Lines of code | $t$ | Halstead time = $e/18$ s |
| $v(g)$ | McCabe cyclomatic complexity | lOCode | Halstead code lines |
| ev$(g)$ | McCabe basic complexity | lOComment | Halstead comment lines |
| iv$(g)$ | McCabe design complexity | lOBlank | Halstead blank line number |
| $n$ | Halstead operands and total number of operators | lOCodeAndComment | Halstead total number of lines of code and comments |
| $v$ | Halstead capacity | uniq_Op | Halstead number of unique operators |
| $l$ | Halstead program length = $(v/n)$ | uniq_Opnd | Halstead number of unique operations |
| $d$ | Halstead difficulty = (1/l) | total_Op | Halstead total number of operators |
| $i$ | Halstead intelligence | total_Opnd | Halstead total number of operators |
| $e$ | Halstead program efficiency = $(v/1)$ | branchCount | Halstead number of branches |
| $b$ | Halstead workload estimate | | |

Table 2: The process metrics for aerospace software.

| Category | Metric | Definition |
|---|---|---|
| Metrics of the software history process | NM | The number of modified modules, which is the number of revisions to the software module before the version is released |
| | NDP | New developer proportion, which is the proportion of new personnel participating in the revision's development |
| | LA | Lines added, that is, the total number of lines of increased code |
| | LD | Lines deleted, that is, the total number of lines deleted code |
| | LM | Lines modified, that is, the total number of lines modified code |
| Metrics of the demand analysis stage in software management | PND | Proportion of new demand |
| | PCD | Proportion of continuing demand |
| | NCD | Number of changes in demand |
| | PDA | Proportion of demand analysts newly joining the project |
| Metrics of the demand design stage in software management | DCR1 | Demand change rate |
| | IDR | The inherit design ratio |
| | PSD1 | Proportion of software designers newly joining the project |
| Metrics of the demand coding stage in software management | DCR2 | Demand change rate |
| | ISD | The inherit ratio of software module |
| | PSD2 | Proportion of new software developers entering the project |

The hidden layer uses sigmoid function as the activation function:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{2}$$

When encoding, use the encoding function $H_n = f(x)(H_n \in [0,1]^m)$ to encode metrics in input layer to get the values of middle layer in the network, where the encoding function is defined as

$$\begin{cases} H_n(x) = f(\varpi x + b), \\ f(x) = \dfrac{1}{1 + \exp(-x)}, \end{cases} \tag{3}$$

where $\varpi$ is the weight matrix and $b$ is the bias vector value.

When decoding, the hidden layer is decoded through the decoding function to get the values of output layer in the network, where the decoding function is defined as

$$x' = G_n(x) = f\left(\varpi^T x + b\right). \tag{4}$$

Among them, $x'$ is the value of output layer ($x' \in [0,1]$); for $f(x)$, the sigmoid function is still used as the activation function for decoding.

The training of the autoencoder is the process of encoding and decoding. In order to achieve the optimization of the training, set the objective function as

$$\frac{1}{n}\sum_{i=1}^{n} L(x_i, x_i'). \tag{5}$$

In the formula, $L(x, x') = \|x - x'\|^2$. In the optimization process, when $x' \approx x$, that is, when the output is as close to the input as possible, the optimal network model will be obtained, which is called single-layer autoencoding network.

*3.2. Deep Neural Network Model.* In the previous section, a single-layer network was constructed through data dimension reduction, but the depth is too shallow, and it will lead to high redundancy of the data feature; thus a good prediction effect that is desired cannot be achieved. Therefore, this paper proposes to use the self-encoding network model to form a deep neural network model to further reduce the dimension and learn the features, which can greatly improve the classification ability and improve the prediction accuracy.

Deep autoencoder [18] is a model of deep learning. The basis of the deep autoencoder network model is autoencoder, which is a network structure (the shape structure of

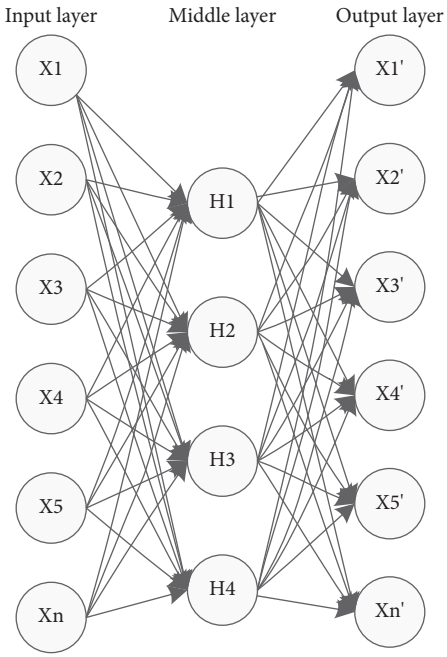


Figure 1: Schematic diagram of automatic encoder.

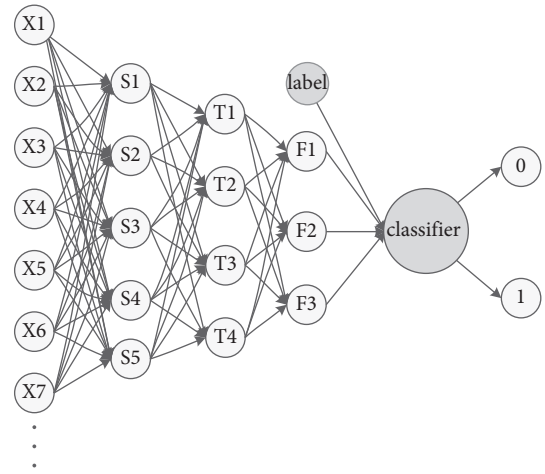


Figure 2: Deep neural network software defect prediction model.

deep autoencoder model) superimposed by multiple autoencoders. The autoencoder is a simple three-layer network structure, which can make the output layer reproduce the data of the input layer as much as possible. Through the first autoencoder, we can get the feature representation of the first layer and then use the features of the first layer as the input of the next autoencoder and train in the same way to get the feature representation of the second layer. Iterating in this way, a deep self-encoding network structure where each layer is a different representation of the original input can be finally gotten. The aerospace software defect prediction model in this article is a deep neural network model that is composed of multiple single-layer autoencoder networks, which takes software defect metrics as the input layer. The middle layer of the previous autoencoding network is no longer connected to the output layer, but is the input of the next autoencoding network, and the dimensions decreased layer by layer. The training mode adopts a layer-by-layer greedy training method to train each layer in turn until the output converge or the input samples are used up. The final weight value and bias vector value achieved are the structural parameters of the deep autoencoding network.

After the training, a classifier is added at the end of the network structure, its classification labels are defective and nondefective, output 1 means defective, and output 0 means nondefective. The deep neural network model constructed in this paper is shown in Figure 2. Using this deep neural network model, data dimension of the $n$ metric features of a sample will be reduced to three dimensions to represent the sample characteristics. There are 31 metric elements in this article; thus $n = 31$. And combined with the data of the historical version of the aerospace software, the training samples are used to continue training through the method of supervised learning until convergence.

When using this model, the feature vector of test sample is put into the outer input layer for calculation, and the classifier will output the probability values of defective and nondefective and classify the result by its probability.

## 4. Experimental Results and Analyses

In this paper, the prediction process of aerospace software engineering is actually as follows (see Figure 3): First, the historical versions of the metrics data and defects data are accumulated through the previous software tests and experimental verifications that have been carried out. Second, the deep neural network training is carried out based on the metrics data and defects data, and the training method adopts the ten-fold crossover method. Finally, the training results are used to predict the defects of the current version of the software. The abovementioned experimental procedure accords with the actual situation of aerospace software engineering development.

In the NASA defect data warehouse, three software projects are selected as the research objects. Table 3, respectively, lists each project's data such as the number of modules, the number of software metrics, and the defect rate of each version.

The experimental data set is trained using the deep neural network model proposed in this research to verify the cross-version software defect prediction ability of the model. The experimental strategies adopted are as follows: use the training set for training and then perform the defects prediction on the test set, each group of experiments is done 10 times, and the results are taken as the average value of the 30 experiments. The experimental groupings are shown in Table 4.

The comparison methods of each grouping training experiment are, respectively, as follows: training with V1.01 for predicting V1.02; training with V1.01 for predicting V1.03; training with V1.02 for predicting V1.03; and training with V1. 01 and V1.02 for predicting V1.03. Finally use confusion matrix to record the trainings. The confusion matrix is shown in Table 5.

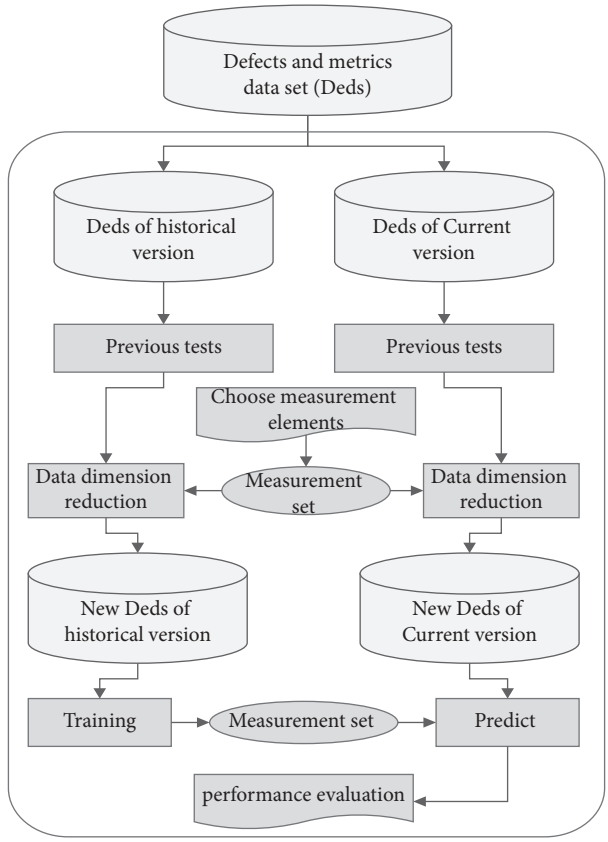


Figure 3: The defect prediction process of aerospace software.

Table 3: Experimental data set.

| Software project | Version number | Number of modules | Number of metrics | Number of defective modules | Defect rate (%) |
|---|---|---|---|---|---|
| XX flight control software | V1.01 | 1837 | 31 | 207 | 11.26 |
| | V1.02 | 1995 | 31 | 215 | 10.77 |
| | V1.03 | 2448 | 31 | 308 | 12.58 |
| XX fire control software | V1.01 | 5973 | 31 | 1218 | 20.39 |
| | V1.02 | 6898 | 31 | 1048 | 15.19 |
| | V1.03 | 8420 | 31 | 1406 | 16.70 |
| XX accused software | V1.01 | 3086 | 31 | 484 | 15.68 |
| | V1.02 | 3317 | 31 | 723 | 21.79 |
| | V1.03 | 3965 | 31 | 511 | 12.89 |

Table 4: Experimental groupings.

| Training set | Prediction set | Result code |
|---|---|---|
| V1.01 | V1.02 | A1-2 |
| V1.01 | V1.03 | A1-3 |
| V1.02 | V1.03 | A2-3 |
| V1.01+V1.02 | V1.03 | A1-2-3 |

Table 5: Confusion matrix.

| | Predicted as defective | Predicted as nondefective |
|---|---|---|
| Actually defective | TP (true positive) | FP (false positive) |
| Actually nondefective | FN (false negative) | TN (true negative) |

Table 6: Statistical results of experimental data.

| Software project | Experimental steps | Accuracy | Recall | Precision | F1-score |
|---|---|---|---|---|---|
| XX flight control software | A1-2 | 0.8843 | 0.7358 | 0.6814 | 0.6767 |
| | A1-3 | 0.8256 | 0.7096 | 0.6722 | 0.6685 |
| | A2-3 | 0.8911 | 0.7504 | 0.7264 | 0.7142 |
| | A1-2-3 | 0.9106 | 0.7826 | 0.7581 | 0.7451 |
| XX fire control software | A1-2 | 0.9372 | 0.8025 | 0.7633 | 0.7562 |
| | A1-3 | 0.9180 | 0.7634 | 0.7052 | 0.6882 |
| | A2-3 | 0.9477 | 0.8175 | 0.7721 | 0.7580 |
| | A1-2-3 | 0.9541 | 0.8260 | 0.7891 | 0.7726 |
| XX accused software | A1-2 | 0.8859 | 0.7246 | 0.6925 | 0.6827 |
| | A1-3 | 0.8663 | 0.7044 | 0.6527 | 0.6475 |
| | A2-3 | 0.8802 | 0.7330 | 0.6992 | 0.6833 |
| | A1-2-3 | 0.8979 | 0.7421 | 0.7143 | 0.7007 |

Subsequently, the experimental results need to be evaluated. The main evaluation indicators in the software defect prediction technology are accuracy, recall, precision, and F1-score. These four evaluation indicators are all calculated based on the confusion matrix [19].

Accuracy refers to the proportion of the correctly predicted numbers of defective and nondefective modules to the total number of modules. This indicator evaluates the effect of the deep neural network model; its calculation formula is as follows:

$$\text{accuracy} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{TP} + \text{FN} + \text{FP}}. \tag{6}$$

Recall, also known as recall rate, refers to the proportion of correctly predicted defective software modules to the number of truly defective modules. Its calculation formula is as follows:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{7}$$

Precision, also known as precision rate, refers to the proportion of correctly predicted defective modules to all predicted defective modules. Its calculation formula is as follows:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{8}$$

F1-score is the harmonic average of recall rate and precision rate. It can be used to comprehensively evaluate the performance of deep neural network models. Its calculation formula is as follows:

$$F1 - \text{score} = \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}. \tag{9}$$

The experimental statistics are shown in Table 6.

Plot the data in the above table according to different software project. The prediction effect of this model is shown in Figure 4.

It can be seen from these figures that the accuracy rate of A-1-2-3 is higher than others, and the accuracy rate of A1-3 is the lowest, which indicates that the prediction effect of adjacent software versions is slightly better than the prediction effect of cross-versions, but lower than the prediction

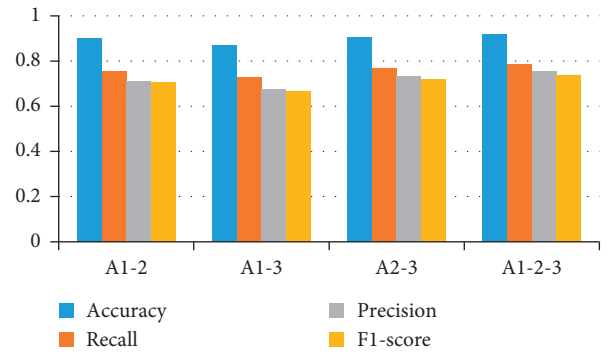


Figure 4: Statistical chart of average data of the three software projects.

effect of multiple versions. Thus, it can be concluded that the changes between two adjacent software versions are usually continuous, so they have better predictive ability; however, if there are more historical versions, the better the predictive effect will be. The data shows that the accuracy of the deep neural network model for aerospace software defect prediction is up to 90%, which has a high prediction effect. However, other indicators are slightly lower, about 75%, which is caused by too few training sets. These indicators of can be improved by increasing the number of training sets.

## 5. Conclusion

By analyzing the characteristics of aerospace software, this paper proposes a software defect prediction method based on process measurement. This study proposed to combine the software process with NASA software data metrics to form a metrics set to design a measurement set oriented to the characteristics of aerospace software. Then, we established a deep neural network through an autoencoder network model to form software defect prediction model. But this method is now only suitable for aerospace engineering software and requires manual testing of the two versions of the data as a training set to achieve better results. Through experimental analysis, the average accuracy of the prediction technology reaches 90%, which proves that the aerospace software defect prediction technology proposed in

this paper is feasible for software defect prediction practical engineering applications and the prediction effect is good.

## Data Availability

(1) Previously reported NASA software defect data sets were used to support this study and are available at DOI: 10.1109/TSE.2013.11. These prior studies are cited at relevant places within the text as references [12]. (2) The process data used to support the findings of this study are included within the article.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## References

[1] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.

[2] Z.-Wu Zhang, *Research on Machine Learning Based Software Defect Prediction*, Nanjing University of Posts and Telecommunications, Jiangsu, Nanjing, China, 2018.

[3] F. Akiyama, "An example of software system debugging," in *Proceedings of the International Federation of Information Processing. Societies Congress*, pp. 353–359, Springer Science and Business Media, New York, Jan. 1971.

[4] M. H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*, Elsevier Science Inc., New York, NY, United States, 1977.

[5] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[6] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Information Sciences*, vol. 179, no. 8, pp. 1040–1058, 2009.

[7] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: a proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[8] H. H. Lu, E. Kocaguneli, and B. Cukic, "Defect prediction between software versions with active learning and dimensionality reduction," in *Proceedings of the International Symposium on Software Reliability Engineering*, pp. 312–322, IEEE, Naples, Italy, Nov. 2014.

[9] X. Yuan, C. Ou, and Y. Wang, "A layer-wise data augmentation strategy for deep learning networks and its soft sensor application in an industrial hydrocracking process," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3296–3305, 2019.

[10] Y. Wang, H. Yang, X. Yuan, Y. A. W. Shardt, C. Yang, and W. Gui, "Deep learning for fault-relevant feature extraction and fault classification with stacked supervised auto-encoder," *Journal of Process Control*, vol. 92, pp. 79–89, 2020.

[11] S. L. Pfleeger, "Software metrics: progress after 25 Years?" *IEEE Software*, vol. 25, no. 6, pp. 32–34, 2008.

[12] T. M. Khoshgoftaar, L. A. Bullard, and K. Gao, "Attribute selection using rough sets in software quality classification," *International Journal of Reliability, Quality and Safety Engineering*, vol. 16, no. 1, pp. 73–89, 2011.

[13] D. Radjenovic, M. Hericko, and R. Torkar, "Software fault prediction metrics: a systematic literature review," *Information and Software Technology*, vol. 55, no. 8, pp. 1397–1418, 2013.

[14] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: some comments on the NASA software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013.

[15] D. Wahyudin, A. Schatten, and D. Winkler, "Defect prediction using combined product and project metrics: a case study from the open source "Apache"MyFaces project family," in *Proceedings of the Software Engineering and Advanced Applications*, pp. 207–215, IEEE, Parma, Italy, Sept. 2008.

[16] J. H. Friedman, "On bias, variance, 0/1-loss, and the curse-of-dimensionality," *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 55–77, 1997.

[17] Y. Bengio, P. Lamblin, and D. Popovici, "Greedy layer-wise training of deep networks," in *Proceedings of the Twenty-First Annual Conference on Neural Information Processing System. NIPS*, ACM, Cambridge, MA, December 2006.

[18] P. Vincent, H. Larochelle, and I. Lajoie, "Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. 3, pp. 3371–3408, 2010.

[19] Mo Zhou, L. Xu, and M. Yang, "Software defect prediction based on deep autoencoder networks," *Computer Engineering & Science*, vol. 40, no. 10, pp. 1796–1804, 2018.