science, engineering and health studies

Research Article

Science, Engineering and Health Studies
https://lio1.tci-thaijo.org/index.php/sehs
ISSN (Online): 2630-0087

Multi-algorithm for predicting the level accuracy of fault output in software

Zulkifli Zulkifli[1], Ford Lumban Gaol[1], Agung Trisetyarso[1], and Widodo Budiharto[2]*

[1] Computer Science Department, Binus Graduate Program, Binus University, Jakarta 11480, Indonesia
[2] Computer Science Department, School of Computer Science, Binus University, Jakarta 11480, Indonesia

ABSTRACT

Software fault output refers to software errors found by testers during the software testing process. Software testing is an overly critical stage in software development; hence, a software testing model is required to systematically classify software errors. For the process of classifying software fault output, accuracy measurements are needed to predict manual fault outputs compared to algorithm-generated fault outputs. Algorithmic methods can be used to measure the accuracy of fault output. The main objective of this research was to compare different algorithms for predicting the accuracy of fault output on a dataset derived from past software testing. The findings indicate that the neural network algorithm outperforms SVM, MLP, RF, and MNB algorithms, achieving 98% accuracy when using 10 software testing variables (function, interface, structure, performance, requirement, documentation, positive, negative, basis path, and times) to predict expected fault outputs.

Keywords: software defect; fault output expected; multi-algorithm

***Corresponding author**:
Zulkifli Zulkifli
zulkifli003@binus.ac.id

***Received**: 18 August 2023*
***Revised**: 31 March 2024*
***Accepted**: 29 April 2024*
***Published**: 2 December 2024*

***Citation**:
Zulkifli, Z., Gaol, F. L., Trisetyarso, A., and Budiharto, W. (2024). Multi-algorithm for predicting the level accuracy of fault output in software. Science, Engineering and Health Studies, 18, 24020006.

1. INTRODUCTION

Software testing has become an activity that incurs significant costs in funding testers, and it also requires a considerable amount of time in the process (Lemos et al., 2018). Discovering software bugs is also a crucial phase in software development (Massoudi et al., 2021).

Fault output refers to software defects found by testers during the software testing phase. Software defect prediction is one of the ways to determine the accuracy level of manual expected fault outputs compared to the fault outputs generated by an algorithm. This helps to maintain the quality of the software (Hammouri et al., 2018).

Using the prediction of potential software defects in software under development helps developers assess the possibility of defects in that software. This approach can save costs and time in the development process.

1.1 Software defect

Software defects are errors, flaws, failures, or faults found during software testing, which result in the developed software not meeting the initial requirements agreed upon with the user (Ali et al., 2023; Thota et al., 2020; Miholca, 2021; Niu et al., 2020).

1.2 Model-based testing (MBT)

MBT is an approach used for testing and predicting errors in a generated model related to software module access (Afzal and Piadehbasmenj, 2021; Ramírez-Méndez et al., 2021). MBT conducts testing based on a previously created system model (Essebaa et al., 2019). This model represents the behavior or functionality of the system under test, and from this model, a series of test cases is generated automatically (Dias-Neto and Travassos, 2009; Petry, 2020). Additionally, it creates a test model by extracting data from the software (El-Far and Whittaker, 2018).

1.3 Model algorithms

In this research, algorithm models are proposed using neural network, support vector machine (SVM), multilayer perceptron (MLP), random forest (RF), and multinomial naïve Bayes (MNB). The neural network algorithm can learn from data by adjusting weights and biases between neurons, allowing it to find patterns that may be difficult to identify by other algorithms. Another advantage is its generalization capability. Although it carries the risk of overfitting, the neural network algorithm can also extract important features from data, enabling generalization to new unseen data (Jean et al., 2022). The SVM algorithm has the advantage of producing optimal solutions to find decision boundaries that maximize the margin between classes and is relatively tolerant to outliers in the dataset (Ali et al., 2023). The MLP algorithm has flexible learning capabilities, where it can learn from nonlinear and unstructured data and can handle complex relationships (Lin et al., 2014). The RF algorithm has resistance to overfitting because each tree is built on a random subset of features and data samples by using techniques such as limiting tree depth, which limits the number of features considered in each split, and aggregating predictions from multiple trees (Masad et al., 2021). The MNB algorithm has the advantage of being efficient in time and memory; it has low time and memory complexity in the training and prediction processes (Hammouri et al., 2018).

1.4 State of the art

Numerous research works have employed algorithms to forecast the precision of defects associated with the expected software fault output. One notable study undertaken by Zulkifli et al., (2022), developed a model-based test (MBT) using 5 variables, including performance, initialization, data structures, incorrect or missing functions, and external database access, with the neural network algorithm. Subsequently, the research was extended by creating an integration-based model (I-BM) framework for MBT. The I-BM framework employed 8 variables: function, interface, structure, performance, requirement, documentation, positive, and negative to predict the accuracy level of its fault output using machine learning approaches (Zulkifli et al., 2023). Furthermore, Ali et al. (2023) conducted research on defect prediction in software using machine learning. Similarly, Dhanda et al. (2022) explored the prediction of software bugs using supervised machine learning algorithms. An improvement made in this work, compared to previous research (Zulkifli et al., 2023), is the addition of two software testing variables. The previous study used 8 variables (function, interface, structure, performance, requirement, documentation, positive, and negative), whereas the current work utilized 10 variables, that is interface (Arcuri, 2020), structure (Luo et al., 2021), function, performance, requirement, documentation (Sonalitha et al., 2020), positive, negative (Zulkifli et al., 2022), basis path, and times). Additionally, this research determined the best algorithm for predicting software defects related to expected fault output.

This research aims to compare the most accurate algorithms in predicting defects, specifically the accuracy level of manual expected fault outputs against fault outputs generated by the algorithms.

2. MATERIALS AND METHODS

2.1 The purpose method

Figure 1 shows the stages of predicting the expected fault output by algorithms. The first stage involved obtaining a dataset with 3 different numbers of variables, namely 5 variables, 8 variables, and 10 variables, which were then used for predicting fault output using the neural network algorithm (Abiodun et al., 2018; Du and Swamy, 2006; Vanmali et al., 2002), SVM (Aydin et al., 2007), MLP (Latif et al., 2021; Naresh et al., 2020), RF (Liu, 2020; Beghoura et al., 2017), and MNB (Catal et al., 2011). After predicting the fault output using the algorithms, the next step involved comparing the accuracy levels of the fault output results generated by each algorithm against the manually expected fault output.

2.2 Data acquisition

To obtain the dataset, software testing was conducted on the digital tracking equipment system (DiTes). The dataset can be accessed at the following Google Drive link: (https://drive.google.com/drive/folders/1LXzJ5y5tMIdCm3w979dsZaul4GTqCFi4?usp=sharing). Table 1 shows the data acquisition results.

Table 1. Data acquisition results

App name	Total testers	Total modules to be tested	Total variables	Total dataset
DiTES	1 person	38 modules	5	190 datasets
			8	304 datasets
			10	380 datasets

This software consists of 38 modules, with each module tested with the 5 variables of function, structure, performance, and requirement, resulting in 190 datasets from testing 38 modules x 5 variables. Furthermore, the datasets generated from software testing with 8 variables (function, interface, structure, performance, requirement, documentation, positive, and negative) tested 38 modules. From this testing, 304 datasets were obtained. In addition, software testing with 10 variables (function, interface, structure, performance, requirement, documentation, positive, negative, times, and basis path) yielded 380 datasets. The results of software testing with 5 variables, 8 variables, and 10 variables yielded a total of 380 datasets.

2.3 Data preprocessing

In this research, preprocessing was carried out using the label encoding technique (Table 2).

Table 2. Results of data preprocessing

Tester	Testing status
1	0
1	0
1	0
1	1

Label encoding is a technique used to change the values in the testing status column to numeric, so that they can be adjusted to the proposed model.

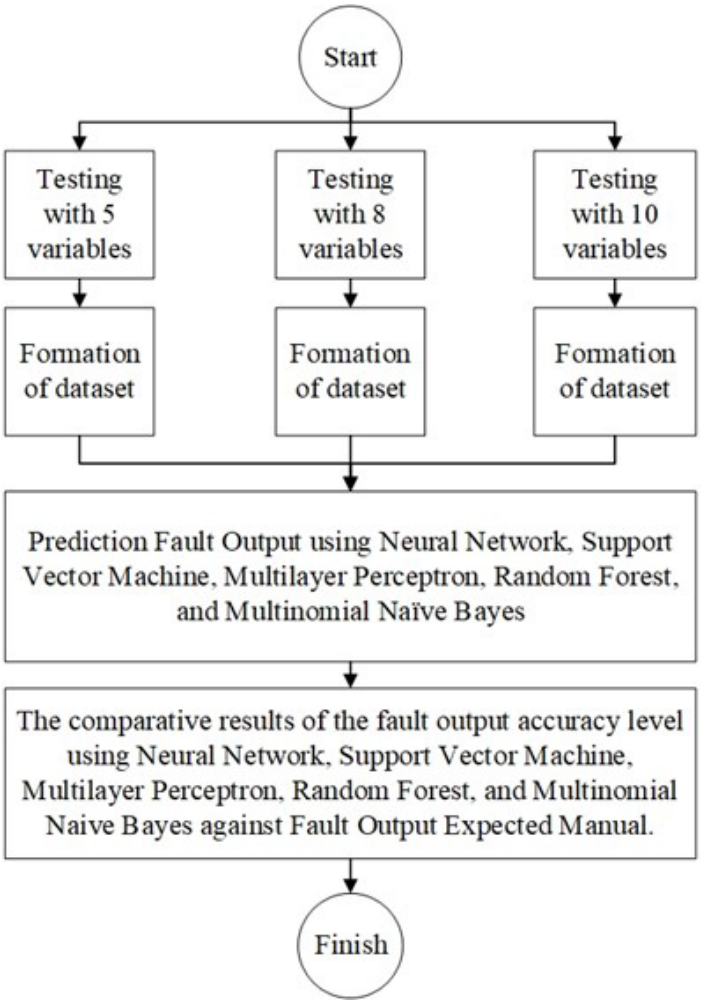

Figure 1. Stages of predicting the expected fault output by algorithms

3. RESULTS

3.1 Model training methodology

3.1.1 Neural network

The stochastic gradient descent (SGD) optimization neural network algorithm model was employed to train each layer with specific configuration parameters, including batch_size=15, epochs=100, and validation_split=0.2, which cover the entire dataset. The outcomes of forecasting the anticipated fault output for each variable using the neural network algorithm are shown in Table 3, and the graphical representation of the predicted fault output for 5 variables using the neural network algorithm is depicted in Figure 2.

Figure 2 shows the prediction results of fault output using 5 variables using the neural network algorithm. In the neural network algorithm, the training accuracy was 0.5 and the training loss was 0.5, while the validation accuracy was 0.71 and the validation loss was 0.48.

Figure 3 shows the prediction results of fault output using 8 variables using the neural network algorithm. In the neural network algorithm, the training accuracy was 0.8 and the training loss was 0.03, while the validation accuracy was 0.8 and the validation loss was 0.01.

Figure 4 shows the prediction results of fault output using 10 variables using the neural network algorithm. The training accuracy was 0.98 and the training loss was 0.01, while the validation accuracy was 1.0 and the validation loss was 0.01.

Table 3. Overall performance of the neural network

Dataset	Epoch	Learning rate	Hidden layer
5 Variable	100	0.2	4
8 Variable	100	0.2	4
10 Variable	100	0.2	4

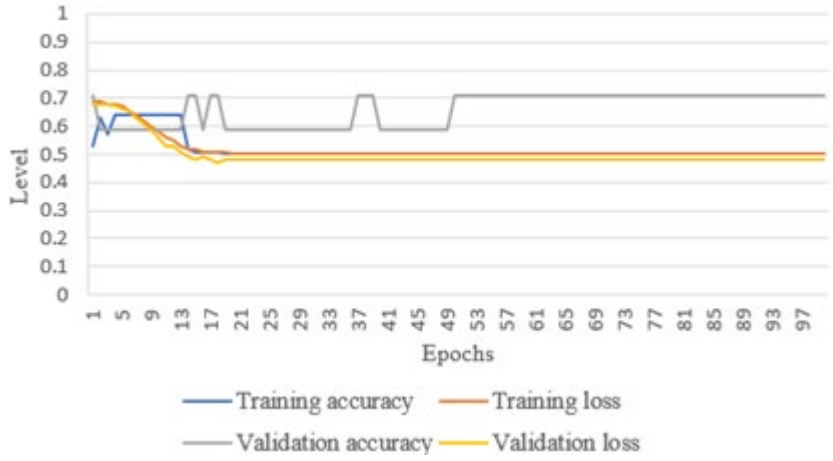

Figure 2. Results of predicting the expected fault output for 5 variables using the neural network algorithm

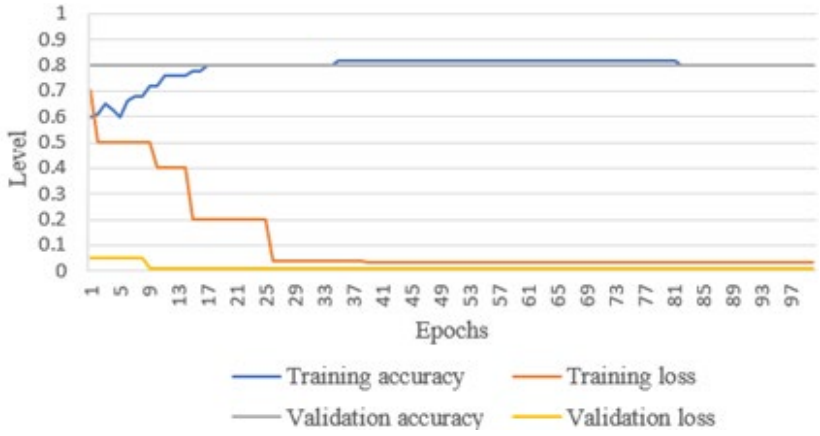

Figure 3. Results of predicting the expected fault output for 8 variables using the neural network algorithm

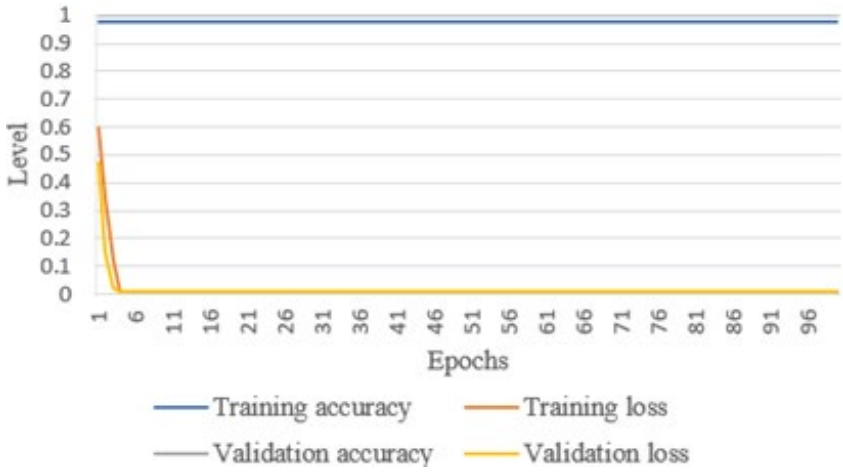

Figure 4. Results of predicting the expected fault output for 10 variables using the neural network algorithm

3.1.2 Support Vector Machine (SVM)

Table 4 shows the results of testing using the SVM algorithm with parameters C=1, kernel='rbf', degree=3, and gamma='scale'. The results of the SVM algorithm, considering all parameters, achieved an accuracy rate of 0.70/70% for 5 variables, 0.82/82% for 8 variables, and 0.97/97% for 10 variables.

3.1.3 Multilayer perceptron (MLP)

Table 5 shows the results of the testing using the MLP algorithm with random_state=1, and max_iter=300. The results of the MLP algorithm shown in Table 2, considering all parameters, achieved an accuracy rate of 0.67/67% for 5 variables, 0.79/79% for 8 variables, and 0.96/96% for 10 variables.

Table 4. Overall performance of SVM

Model	Dataset	Parameters	Accuracy
SVM	5 Variable	(rbf; C=1; degree=3; scale)	0.70
	8 Variable	(rbf; C=1; degree=3; scale)	0.82
	10 Variable	(rbf; C=1; degree=3; scale)	0.97

Table 5. Overall performance of MLP

Model	Dataset	Parameters	Accuracy
MLP	5 Variable	(random_state=1, max_iter=300)	0.67
	8 Variable	(random_state=1, max_iter=300)	0.79
	10 Variable	(random_state=1, max_iter=300)	0.96

3.1.4 Random forest (RF)

Table 6 shows the results of the testing using the RF algorithm with parameters max_depth=5, random_state=10, n_estimators=500, and criterion='entropy'.

The results of the RF algorithm shown, considering all parameters, achieved an accuracy rate of 0.50/50% for 5 variables, 0.78/78% for 8 variables, and 0.95/95% for 10 variables.

3.1.5 Multinomial Naïve Bayes (MNB)

Table 7 shows the results of the testing using the MNB model with parameters alpha=2.0, force_alpha='warn', fit_prior=true, and class_prior=none. The results of the MNB algorithm, considering all parameters, achieved an accuracy rate of 0.66/66% for 5 variables, 0.76/76% for 8 variables, and 0.96/96% for 10 variables.

Table 6. Overall performance of RF

Model	Dataset	Parameters	Accuracy
RF	5 variables	(max_depth=5, random_state=10, n_estimators=500, criterion='entropy')	0.50
	8 variables	(max_depth=5, random_state=10, n_estimators=500, criterion='entropy')	0.78
	10 variables	(max_depth=5, random_state=10, n_estimators=500, criterion='entropy')	0.95

Table 7. Overall performance of MNB

Model	Dataset	Parameters	Accuracy
MNB	5 variables	(alpha=2.0, force_alpha='warn', fit_prior=true, and class_prior=none)	0.66
	8 variables	(alpha=2.0, force_alpha='warn', fit_prior=true, and class_prior=none)	0.76
	10 variables	(alpha=2.0, force_alpha='warn', fit_prior=true, and class_prior=none)	0.96

3.2 Evaluation confusion matrix

The proposed model was evaluated using a confusion matrix, including accuracy, precision, recall, and F1-score (Ruuska et al., 2018). The proposed model was evaluated using 61 software testing datasets derived from 5 variables, 8 variables, and 10 variables.

3.2.1 Neural network

Figure 5 displays the output of the confusion matrix from the neural network algorithm for 5 variables, 8 variables, and 10 variables. In the 5-variable dataset, there were 41 successfully and 20 unsuccessfully classified validation classes out of 61 available data. For the 8-variable dataset, there were 50 successfully and 11 unsuccessfully classified validation classes out of 61 available data. Meanwhile, in the 10-variable dataset, there were 60 successfully and 1 unsuccessfully classified validation classes out of 61 available data. Tables 8–10 present the evaluation results of the neural network algorithm for the 5-variable, 8-variable, and 10-variable datasets.

3.2.2 Support vector machine (SVM)

Figure 6 illustrates the output of the confusion matrix from the support vector machine algorithm for 5 variables, 8 variables, and 10 variables. In the 5-variable dataset, there were 46 successfully and 15 unsuccessfully classified validation classes out of 61 available data. For the 8-variable dataset, there were 51 successfully and 10 unsuccessfully classified validation classes out of 61 available data. In addition, in the 10-variable dataset, there were 60 successfully and 1 unsuccessfully classified validation classes out of 61 available data. Tables 11–13 present the evaluation results of the support vector machine algorithm for the 5-variable, 8-variable, and 10-variable datasets.

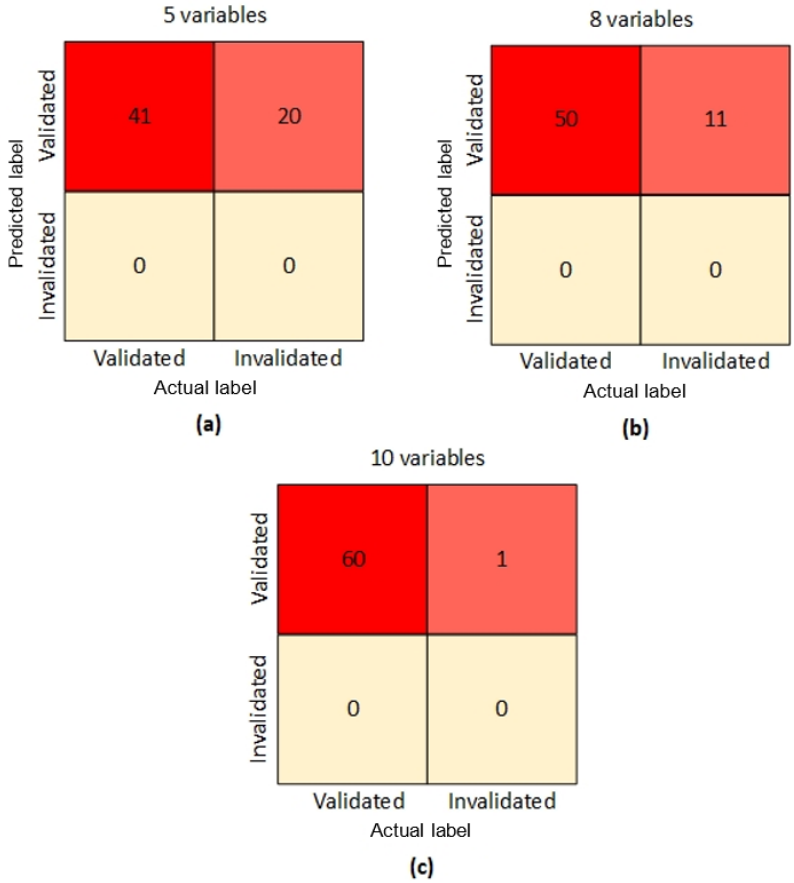

Figure 5. Results of the confusion matrix of the neural network algorithm on fault output for (a) 5 variables, (b) 8 variables, and (c) 10 variables

Table 8. Results of the performance evaluation of the neural network algorithm model for a dataset of 5 variables

	Precision	**Recall**	**F1-score**	**Class**
	1.00	0.50	0.67	Validated
	0.57	1.00	0.73	Invalidated
Accuracy	0.70			

Table 9. Results of the performance evaluation of the neural network algorithm model for a dataset of 8 variables

	Precision	**Recall**	**F1-score**	**Class**
	1.00	0.60	0.77	Validated
	0.67	1.00	0.83	Invalidated
Accuracy	0.80			

Table 10. Results of the performance evaluation of the neural network algorithm model for a dataset of 10 variables

	Precision	**Recall**	**F1-score**	**Class**
	0.98	1.00	0.99	Validated
	1.00	0.98	0.99	Invalidated
Accuracy	0.98			

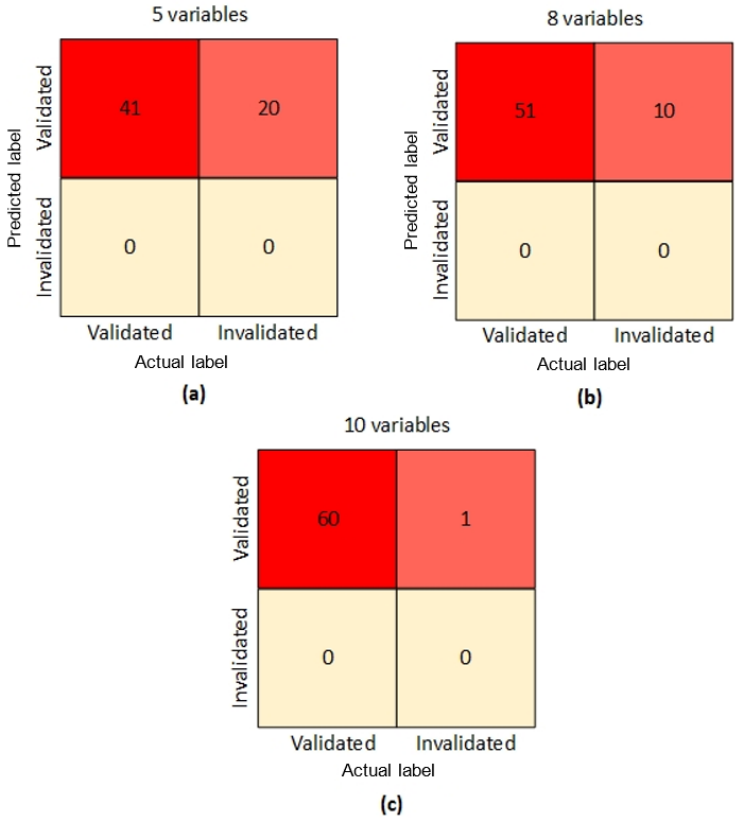

Figure 6. Results of the confusion matrix of the SVM algorithm on fault output for (a) 5 variables, (b) 8 variables, and (c) 10 variables

Table 11. Results of the performance evaluation of the SVM algorithm model for a dataset of 5 variables

	Precision	Recall	F1-score	Class
	1.00	0.50	0.67	Validated
	0.57	1.00	0.73	Invalidated
Accuracy	0.70			

Table 12. Results of the performance evaluation of the SVM algorithm model for a dataset of 8 variables

	Precision	Recall	F1-score	Class
	1.00	0.62	0.72	Validated
	0.69	1.00	0.85	Invalidated
Accuracy	0.82			

Table 13. Results of the performance evaluation of the SVM algorithm model for a dataset of 10 variables

	Precision	Recall	F1-score	Class
	0.98	1.00	0.98	Validated
	1.00	0.98	0.97	Invalidated
Accuracy	0.97			

3.2.3 Multilayer perceptron (MLP)

Figure 7 displays the output of the confusion matrix from the multilayer perceptron algorithm for (a) 5 variables, (b) 8 variables, and (c) 10 variables. In the 5-variable dataset, there were 40 successfully and 21 unsuccessfully classified validation classes out of 61 available data. For the 8-variable dataset, there were 49 successfully and 12 unsuccessfully classified validation classes out of 61 available data. In addition, in the 10-variable dataset, there were 60 successfully and 1 unsuccessfully classified validation classes out of 61 available data. Tables 14–16 present the evaluation results of the multilayer perceptron algorithm for the 5-variable, 8-variable, and 10-variable datasets.

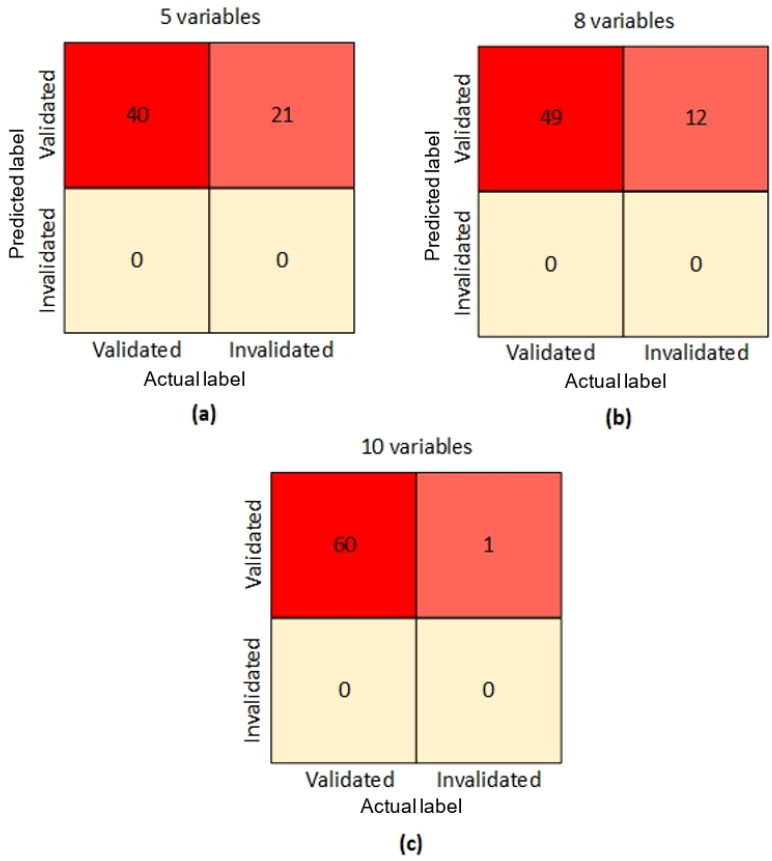

Figure 7. Results of the confusion matrix of the MLP algorithm on fault output for (a) 5 variables, (b) 8 variables, and (c) 10 variables

Table 14. Results of the performance evaluation of the MLP algorithm model for a dataset of 5 variables

	Precision	Recall	F1-score	Class
	0.98	0.50	0.65	Validated
	0.57	0.98	0.71	Invalidated
Accuracy	0.67			

Table 15. Results of the performance evaluation of the MLP algorithm model for a dataset of 8 variables

	Precision	Recall	F1-score	Class
	0.97	0.62	0.69	Validated
	0.68	0.98	0.82	Invalidated
Accuracy	0.79			

Table 16. The results of the performance evaluation of the neural MLP model for a dataset of 10 variables

	Precision	Recall	F1-score	Class
	0.98	0.99	0.97	Validated
	0.99	0.98	0.97	Invalidated
Accuracy	0.96			

3.2.4 Random Forest (RF)

Figure 8 depicts the output of the confusion matrix from the random forest algorithm for 5 variables, 8 variables, and 10 variables. In the 5-variable dataset, there were 29 successfully and 32 unsuccessfully classified validation classes out of 61 available data. For the 8-variable dataset, there were 49 successfully and 12 unsuccessfully classified validation classes out of 61 available data. In addition, in the 10-variable dataset, there were 60 successfully and 1 unsuccessfully classified validation classes out of 61 available data. Tables 17–19 showcase the evaluation results of the RF algorithm for the 5-variable, 8-variable, and 10-variable datasets.

3.2.5 Multinomial Naïve Bayes (MNB)

Figure 9 illustrates the output of the confusion matrix from the MNB algorithm for 5 variables, 8 variables, and 10 variables. In the 5-variable dataset, there were 40 successfully and 21 unsuccessfully classified validation classes out of 61 available data. For the 8-variable dataset, there were 49 successfully and 12 unsuccessfully classified validation classes out of 61 available data. In addition, in the 10-variable dataset, there were 60 successfully and 1 unsuccessfully classified validation classes out of 61 available data. Tables 20–22 present the evaluation results of the MNB algorithm for the 5-variable, 8-variable, and 10-variable datasets.

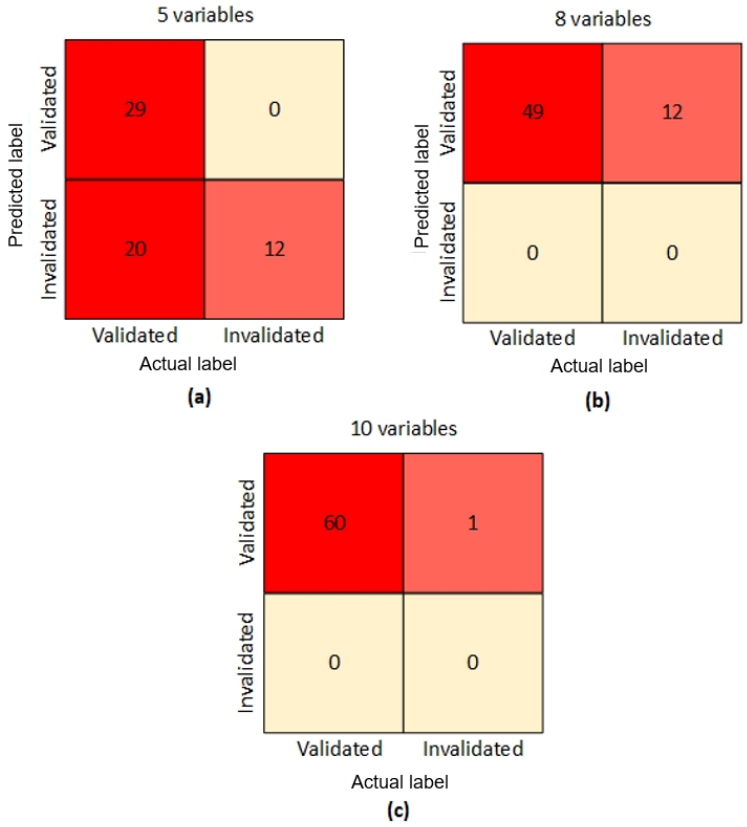

Figure 8. Results of the confusion matrix of the RF algorithm on fault output for (a) 5 variables, (b) 8 variables, and (c) 10 variables

Table 17. Results of the performance evaluation of the RF algorithm model for a dataset of 5 variables

	Precision	Recall	F1-score	Class
	1.00	0.29	0.44	Validated
	0.38	1.00	0.55	Invalidated
Accuracy	0.50			

Table 18. Results of the performance evaluation of the RF algorithm model for a dataset of 8 variables

	Precision	Recall	F1-score	Class
	0.97	0.62	0.61	Validated
	0.69	0.97	0.82	Invalidated
Accuracy	0.78			

Table 19. Results of the performance evaluation of the RF algorithm model for a dataset of 10 variables

	Precision	Recall	F1-score	Class
	0.97	0.96	0.97	Validated
	0.96	0.94	0.95	Invalidated
Accuracy	0.95			

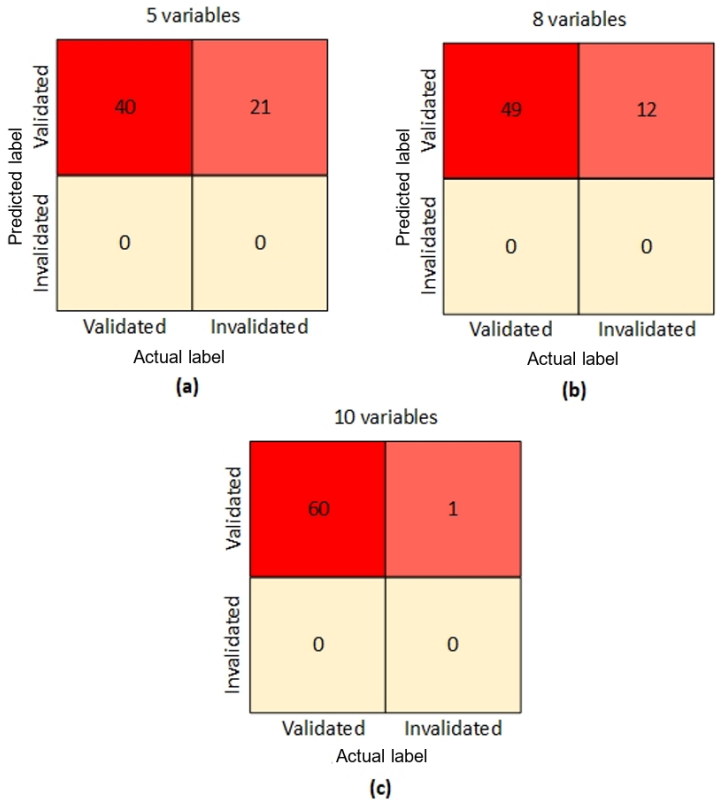

Figure 9. Results of the confusion matrix of the MNB algorithm on fault output for (a) 5 variables, (b) 8 variables, and (c) 10 variables

Table 20. Results of the performance evaluation of the MNB algorithm model for a dataset of 5 variables

	Precision	Recall	F1-score	Class
	0.98	0.50	0.65	Validated
	0.57	0.85	0.69	Invalidated
Accuracy	0.66			

Table 21. Results of the performance evaluation of the MNB algorithm model for a dataset of 8 variables

	Precision	Recall	F1-score	Class
	0.98	0.50	0.65	Validated
	0.57	0.85	0.69	Invalidated
Accuracy	0.66			

Table 22. Results of the performance evaluation of the MNB algorithm model for a dataset of 5 variables

	Precision	Recall	F1-score	Class
	0.98	0.99	0.97	Validated
	0.99	0.98	0.97	Invalidated
Accuracy	0.96			

3.3 Comparison of the fault output of neural network algorithms, SVM, MLP, RF, and MNB against the expected actual error output

Figure 10 depicts the accuracy curve of the neural network algorithm compared to the actual expected values. Figure 11 showcases the accuracy curve of the SVM algorithm in relation to the actual expected values. The accuracy curve of the MLP algorithm, RF algorithm, and MNB algorithm relative to the actual expected values are shown in Figures 12, 13, and 14, respectively.

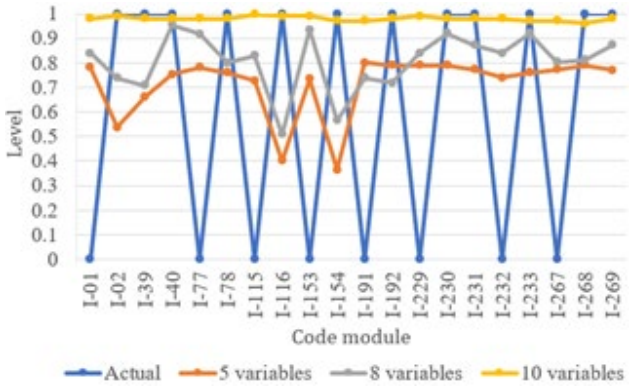

Figure 10. Accuracy level of the neural network algorithm on the actual expected fault output

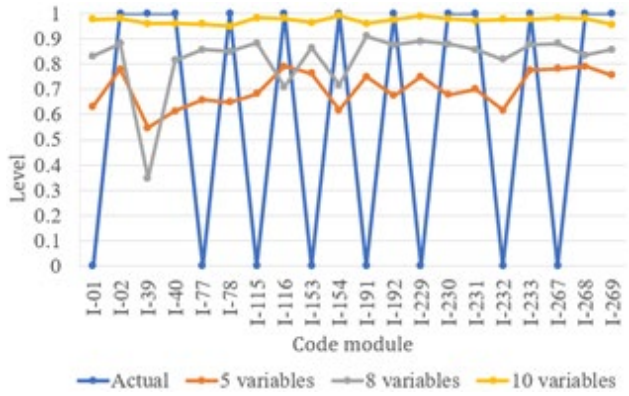

Figure 11. Accuracy level of the SVM algorithm on the actual expected fault output

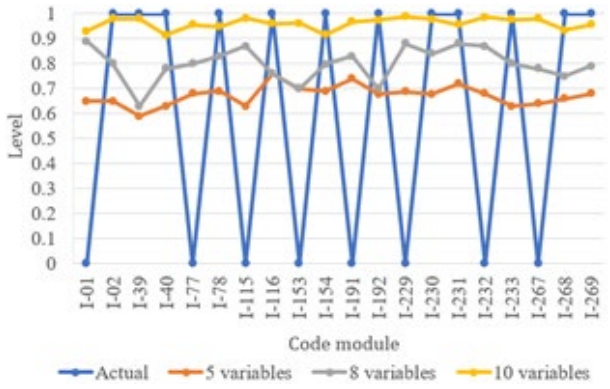

Figure 12. Accuracy level of the MLP algorithm on the actual expected fault output

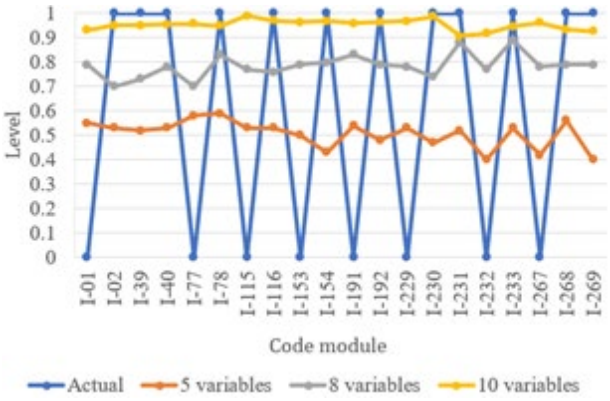

Figure 13. Accuracy level of the RF algorithm on the actual expected fault output

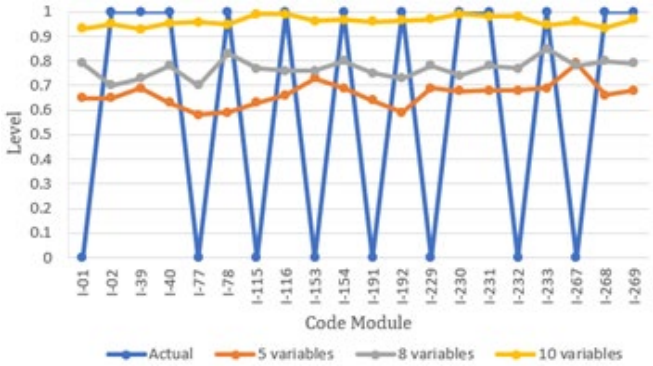

Figure 14. Accuracy level of the MNB algorithm on the actual expected fault output

Figures 10 – 14 depict the comparison of fault output between the neural network, SVM, MLP, RF, and MNB algorithms against the actual expected values. The neural network algorithm achieved average accuracies of 0.71, 0.80, and 0.98 for each variable, respectively. The SVM algorithm obtained average accuracies of 0.70, 0.82, and 0.97 for each variable, respectively. The MLP algorithm achieved average accuracies of 0.67, 0.79, and 0.96 for each variable, respectively. The RF algorithm obtained average accuracies of 0.50, 0.78, and 0.95 for each variable, respectively. The MNB algorithm achieved average accuracies of 0.66, 0.76, and 0.96 for each variable, respectively.

3.4 Practical complexity/application of I-bM framework software testing assessment tools

At this stage, estimating effort is done through the use case point (UCP) method. Effort estimation serves as the basis for estimating time, human resources, and costs, as shown in Tables 23–24. The effort calculation begins with the value of unadjusted actor weight (UAW). UAW is calculated based on the actors involved in the system mapped against actor categories to obtain the weight of each actor.

After obtaining the actor weights and the use case weights, the calculation of unadjusted actor weight (UAW) and unadjusted use case weight (UUCW) were then carried out, as shown in Tables 25–26.

Table 23. Weight value for actor

Number	Actor type	Weight
1.	Simple	1
2.	Average	2
3.	Complex	3

Table 24. Weight value for use case

Number	Actor type	Weight
1.	Simple	5
2.	Average	10
3.	Complex	15

Table 25. Calculation of the UAW value

Number	Actor type	Weight	Number of actors	Amount
1.	Simple	1	0	0
2.	Average	2	1	2
3.	Complex	3	1	3
Total UAW				**5**

Table 26. Calculation of the UUCW value

Number	Actor type	Weight	Number of actors	Amount
1.	Simple	5	0	0
2.	Average	10	4	40
3.	Complex	15	2	30
Total UAW				**70**

The next step was to calculate the value of unadjusted use case points (UUCP), which was obtained by summing the total value of UAW and the total value of UUCW.

$$UUCP = UUCW + UAW$$

$$UUCP = 70 + 5 = 75$$

The technical complexity factor (TCF) and the environment complexity factor (ECF) were then calculated, as shown in Tables 27–28.

Table 27. Calculation of the TCF value

Number	Technical factor	Weight	Score	Weight * Score
T1	Distributed system required	2	4	8
T2	Response time	2	4	8
T3	End user efficiency	1	4	4
T4	Complex internal processing required	1	4	4
T5	Reusable code	1	4	4
T6	Easy to install	0.5	5	2.5
T7	Easy to use	0.5	5	2.5
T8	Portable	2	5	10
T9	Easy to change	1	5	5
T10	Concurrent	1	3	3
T11	Security features	1	4	4
T12	Access for third parties	1	4	4
T13	Special training required	1	4	4
Total TF				**63**
Total TCF (0.6+(0.01 * TF))				**1.23**

Table 28. Calculation of the ECF value

Number	Technical factor	Weight	Score	Weight * Score
E1	Familiarity with the project	1.5	4	6
E2	Application experience	0.5	5	2.5
E3	OO-programming experience	1	2	2
E4	Lead analyst capability	0.5	5	2.5
E5	Motivation	1	5	5
E6	Stable requirements	2	5	10
E7	Part time staff	-1	3	-3
E8	Difficult programming language	-1	5	-5
Total EF				**20**
Total ECF (1.4+(-0.03 * EF))				**0.8**

After obtaining the values of TCF and ECF, the next step was to calculate the use case points (UCP) and hours of effort.

$$UCP = UUCP * TCF * ECF$$

$$UCP = 75 * 1.25 * 0.8 = 73.8$$

*Hours of effort = UCP * 20 = 73.8 * 20 = 1476 (man-hours)*

The results of UCP and hours of effort were 73.8 and 1476 (man-hours) respectively. The estimated time and cost that will be used in the development of the I-bM framework application were then calculated, as shown in Tables 29–32.

Table 29. Practical complexity of application of IbM framework assessment tools

Effort	**1476**	**Productivity factor**			
Productivity factor (Karner, 1993), PF	**28**	**20**	**10**	**5**	**4**	**2**
Duration (hours)	41,328	29,520	14,760	7,380	5,904	2,952
Duration (weeks)	861	615	307.5	153.75	123	61.5
Duration (months)	112.92	80.66	40.33	20.16	16.13	8.07

Table 30. Calculation of the estimated time and costs with PF=4

Number	Position	Amount	Workload (hours/day)	Workload (hours/week)	Total (hours/week)	Workload total	Hourly salary (Rp.)	Sum salary
1	System analyst	1	4	20	20	2,460	20.000	49.200.000
2	Programmer	1	8	40	40	4,920	20.000	98.400.000
3	Tester	1	4	20	20	2,460	20.000	49.200.000
Total		**3**						**196.800.000**

Table 31. Calculation of the estimated time and costs with PF=5

Number	Position	Amount	Workload (hours/day)	Workload (hours/week)	Total (hours/week)	Workload total	Hourly salary (Rp.)	Sum salary
1	System analyst	1	4	20	20	3,075	20.000	61.500.000
2	Programmer	1	8	40	40	6,150	20.000	123.000.000
3	Tester	1	4	20	20	3,075	20.000	61.500.000
Total		**3**						**246.000.000**

Table 32. Calculation of the estimated time and costs with PF=10

Number	Position	Amount	Workload (hours/day)	Workload (hours/week)	Total (hours/week)	Workload total	Hourly salary (Rp.)	Sum salary
1	System analyst	1	4	20	20	6,150	20.000	123.000.000
2	Programmer	1	8	40	40	12,300	20.000	246.000.000
3	Tester	1	4	20	20	6,150	20.000	123.000.000
Total		**3**						**492.000.000**

From the results of estimating time and cost with PF = 4, 5, and 10, it is shown that in the development of the application until the software testing phase, the cost is very expensive, requiring time and human resources. Practical complexity and application tools assessment of software testing in the I-bM framework indicated the best cost, time estimation, and human resources: a cost of Rp.196,800,000, with an estimated time of 9,840 and a total of 3 personnels.

4. DISCUSSION

The results of experiments conducted for 5, 8, and 10 variables indicate that the neural network algorithm outperforms other algorithms. This is because the neural network algorithm can learn from data by adjusting weights and biases between neurons, enabling it to discover patterns that may be difficult to identify by other algorithms. Another advantage is its generalization capability; although it carries the risk of overfitting, the neural network algorithm also has the ability to extract important features from data, allowing for generalization to new unseen data. The findings of this research could serve as a reference for software testing models and the implementation of neural networks, SVM, MLP, RF, and MNB algorithms in measuring the accuracy level of software fault output.

5. CONCLUSION

The results of this study, after conducting experiments on datasets generated from software testing with 5 variables, 8 variables, and 10 variables, show the average accuracy of the neural network algorithm for fault output is expected to be 0.71, 0.80, and 0.98; with the average accuracy using SVM of 0.70, 0.82, and 0.97; the average accuracy using MLP of 0.67, 0.79 and 0.96; the average accuracy using RF of 0.50, 0.78, and 0.95; and the average accuracy using MNB of 0.66, 0.76, and 0.96 against the manually expected fault output. From the above results, it can be concluded that the neural network algorithm with 10 variables in software testing has the highest accuracy for fault output expected with an average value of 0.98.

REFERENCES

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A. E., and Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11), e00938.

Afzal, W., and Piadehbasmenj, A. (2021). Cloud-based architectures for model-based simulation testing of embedded software. In *Proceedings of the 10th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–8. Budva, Montenegro.

Ali, S. A., Roy, N. R., and Raj, D. (2023). Software defect prediction using machine learning. In *Proceedings of the 10th International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 639–642. New Delhi, India.

Arcuri, A. (2020). Automated black-and white-box testing of RESTful APIs with EvoMaster. *IEEE Software*, 38(3), 72–78.

Aydin, I., Karaköse, M., and Akin, E. (2007). Artificial immune based support vector machine algorithm for fault diagnosis of induction motors. In *Proceedings of the International Aegean Conference on Electrical Machines and Power Electronics*, pp. 217–221. Bodrum, Turkey.

Beghoura, M. A., Boubetra, A., and Boukerram, A. (2017). Green software requirements and measurement: Random decision forests-based software energy consumption profiling. *Requirements Engineering*. 22(1), 27–40.

Catal, C., Alan, O., and Balkan, K. (2011). Class noise detection based on software metrics and ROC curves. *Information Sciences*, 181(21), 4867–4877.

Dhanda, N., Datta, S. S., and Dhanda, M. (2022). Machine learning algorithms. In *Research Anthology on Machine Learning Techniques, Methods, and Applications* (Khosrow-Pour, M., Ed.), pp. 849–869. Pennsylvania: IGI Global.

Dias-Neto, A. C., and Travassos, G. H. (2009). Model-based testing approaches selection for software projects. *Information and Software Technology*, 51(11), 1487–1504.

Du, K.-L., and Swamy, M. N. S. (2006). Neural Networks in a Softcomputing Framework, London: Springer. pp. 1–566.

El-Far, I. K., and Whittaker, J. A. (2018). Model-based software testing. In *Encyclopedia of Software Engineering* (Marciniak, J. J., Ed.), pp. 1–22. New Jersey: John Wiley & Sons.

Essebaa, I., Chantit, S., and Ramdani, M. (2019). Model-based testing from model driven architecture: A novel approach for automatic test cases generation. In *Advances in Smart Technologies Applications and Case Studies. SmartICT 2019: Lecture Notes in Electrical Engineering, vol 684* (El Moussati, A., Kpalma, K., Ghaouth Belkasmi, M., Saber, M., Guégan, S., Eds.), pp. 600–609. Cham: Springer.

Hammouri, A., Hammad, M., Alnabhan, M. M., Alnabhan, M., and Alsarayrah, F. (2018). Software bug prediction using machine learning approach. *International Journal of Advanced Computer Science and Applications*, 9(2), 78–83.

Jean, A. K., Diarra, M., Bakary, B. A., Pierre, G., and Jérôme, A. K. (2022). Application based on hybrid CNN-SVM and PCA- SVM approaches for classification of cocoa beans. *International Journal of Advanced Computer Science and Applications*, 13(9), 231–238.

Latif, A., Fitriana, L. A., and Firdaus, M. R. (2021). Comparative analysis of software effort estimation using data mining technique and feature selection. *Journal of Computer Science and Technology*, 6(2), 167–174.

Lemos, O. A. L., Silveira, F. F., Ferrari, F. C., and Garcia, A. (2018). The impact of software testing education on code reliability: An empirical assessment. *Journal of Systems and Software*, 137, 497–511.

Lin, M., Chen, Q., and Yan, S. (2014). Network in network. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR) 2014*, pp. 1–10. Banff, Canada.

Liu, S. (2020). An integrated scheme for online dynamic security assessment based on partial mutual information and iterated random forest. *IEEE Transactions on Smart Grid*, 11(4), 3606–3619.

Luo, R., Huang S., Chen, H., and Chen M. (2021). Code confusion in white box crowdsourced software testing. *International Journal of Performability Engineering*, 17(3), 276–288.

Masad, I. S., Alqudah, A., Alqudah, A. M., and Almashaqbeh, S. (2021). A hybrid deep learning approach towards building an intelligent system for pneumonia detection in chest x-ray images. *International Journal of Electrical and Computer Engineering*, 11(6), 5530–5540.

Massoudi, M., Jain, N. K., and Bansal, P. (2021). Software defect prediction using dimensionality reduction and deep learning. In *Proceedings of the 3rd International Conference on Intelligent Communication Technologies and Virtual Mobile Networks*, pp. 884–893. Tirunelveli, India.

Miholca, D. L. (2021). New conceptual cohesion metrics: Assessment for software defect prediction. In *Proceedings of the 23rd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 163–170. Timisoara, Romania.

Naresh, E., Naik, M. D., Niranjanamurthy, M., and Shankar, S. P. (2020). Functional testing on heart disease detection using multilayer perceptron technique. *Journal of Computational and Theoretical Nanoscience,* 17(9–10), 3915–3920.

Niu, W., Zhang, X., Du, X., Zhao, L., Cao, R., and Guizani, M. (2020). A deep learning based static taint analysis approach for IoT software vulnerability location. *Measurement*, 152, 107139.

Petry, K. L. (2020). Model-based testing of software product lines: Mapping study and research roadmap. *Journal of Systems and Software*, 167, 110608.

Ramírez-Méndez, J., Quesada-López, C., Martínez, A., and Jenkins, M. (2021). Agent-oriented approaches for model-based software testing: A mapping study. In *Advances in Intelligent Systems and Computing* (Rocha, Á., Ferrás, C., López-López, P. C., and Guarda, T., Eds.), pp. 340–349. Cham: Springer.

Ruuska, S., Hämäläinen, W., Kajava, S., Mughal, M., Matilainen, P., and Mononen, J. (2018). Evaluation of the confusion matrix method in the validation of an automated system for measuring feeding behaviour of cattle. *Behavioural Processes*, 148, 56–62.

Sonalitha, E., Nurdewanto, B., Zubair, A., Asriningtias, S. R., Yudhistiro, K., and Mujahidin, I. (2020). Blackbox testing model boundary value of mapping taxonomy applications and data analysis of art and artworks. In *Proceedings of the 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, pp. 7–11. Yogyakarta, Indonesia.

Thota, M. K., Shajin, F. H., and Rajesh, P. (2020). Survey on software defect prediction techniques. *International Journal of Applied Science and Engineering*, 17(4), 331–344.

Vanmali, M., Last, M., and Kandel, A. (2002). Using a neural network in the software testing process. *International Journal of Intelligent Systems*, 17(1), 45–62.

Zulkifli, Gaol, F. L., Trisetyarso, A., and Budiharto, W. (2022). Software testing model by measuring the level of accuracy fault output using neural network algorithm. In *Proceedings of the 2022 IEEE 7th International Conference on Information Technology and Digital Applications (ICITDA)*, pp. 1–6. Yogyakarta, Indonesia.

Zulkifli, Z., Gaol, F. L., Trisetyarso, A., and Budiharto, W. (2023). Software testing integration-based model (I-BM) framework for recognizing measure fault output accuracy using machine learning approach. *International Journal of Software Engineering and Knowledge Engineering*, 33(08), 1149–1168.