Transaction on Informatics and Data Science
Vol. 1, No. 1, 2024, 11-20
ISSN: 3064-1772 (online)
DOI: https://doi.org/10.24090/tids.v1i1.12192

Naive Bayes Classification for Software Defect Prediction

Edwin Hari Agus Prastyo*[1], Muhammad Ainul Yaqin [1], Suhartono [1], M. Faisal [1], Reza Augusta Jannatul Firdaus [2]

[1] Department of Informatics, Faculty of Science and Technology, Universitas Islam Negeri Maulana Malik Ibrahim, Malang, Indonesia.

[2] Department of Informatics, Faculty of Information Technology, Universitas Hasyim Asy'Ari, Jombang, Indonesia.

Article Information

Submitted February 2, 2024
Accepted March 10, 2024
Published April 1, 2024

Abstract

Software defects are an inevitable aspect of software development, exerting substantial influence on the reliability and performance of software applications. This research addresses the imperative need to enhance the prediction and monitoring of software defects within the software development domain. With a focus on system stability and the prevention of software malfunctions, this study underscores the significance of proactive measures, including robust software testing, routine maintenance, and continuous system monitoring. The central challenge addressed in this research pertains to the insufficient efficiency of predicting software defects during the development phase. To address this challenge, the study employs the Naive Bayes classification method. Test results conducted on the complete dataset reveal that the Naive Bayes method yields classifications with an exceptionally high accuracy rate, reaching 98%. These findings suggest that the method holds great potential as an effective tool for predicting and preventing software defects throughout the software development process. Additionally, through linear regression analysis, the model exhibits an intercept value of -0.09359968 and a coef coefficient of 0.00761893. The outcomes of this research bear significant implications for the implementation of the Naive Bayes method in software bug prediction analysis, particularly in the utilization of the Python programming language with the assistance of Google Colab. The adoption of this method can play a pivotal role in mitigating risks and elevating the overall quality of software during the developmental stages.

Keywords: Software flaw prediction, software defect prediction, Naïve Bayes

1. Introduction

In an era of rapidly developing technology, software defect prediction has become a crucial step in identifying and mitigating potential problems in software development. One approach that has been analyzed in depth is the application of Naive Bayes classification to improve software defect predictions [1].

The main problem to be resolved in this research is the lack of efficient predictions related to software defects in software development. Continuous monitoring and risk management are important aspects of software development and maintenance. Effective risk management involves early identification and analysis of risks, implementation of corrective actions, continuous monitoring, and reassessment [2]. This process is important for reducing unexpected defects and system crashes, ensuring data integrity, and improving user experience and application reliability [3]. Continuous monitoring is critical to identifying major vehicle defects and reducing accidents resulting from vehicle-related impairments. [4]. It also plays an important role in improving the safety of heavy vehicles by reducing defect-related accidents through inspections and maintenance programs [5].

In the field of risk management, the use of Bayesian Networks has been proposed to

* **Author Correspondence**: Edwin Hari Agus Prastyo: Universitas Islam Negeri Maulana Malik Ibrahim, Jalan Gajayana No. 50 Malang 65144, Jawa Timur - Indonesia. Email: 220605220005@student.uin-malang.ac.id

support decision making in various software designs, contributing to the effective management of technological risks in software projects [2]. This shows the importance of leveraging advanced techniques and methodologies to address risks in development and maintenance.

The Naive Bayes classification method has been widely used in various fields, including software error prediction, sentiment analysis, and disease prediction. In the field of software error prediction, researchers have applied the Naive Bayes classifier to predict software errors using techniques such as sampling and feature selection [6], integration of distribution-based balance and ensemble bagging [7], and comparison with other classification algorithms. [8]. This study has demonstrated the effectiveness of the Naive Bayes classifier in predicting software defects. In the field of feelings analysis, the Naive Bayes classifier has been used to analyze people's feelings on social media platforms such as Twitter [9]. It has been applied to analyze sentiment during the 2020 election in the context of the COVID-19 pandemic. The results of this study demonstrate the applicability of the Naive Bayes classifier in sentiment analysis tasks. In addition, the Naive Bayes classifier has also been used in disease prediction. For example, in the healthcare field, Naive Bayes classifiers have been used to identify diseases such as tuberculosis [5] and gingivitis [10]. This study highlights the accuracy and effectiveness of the Naive Bayes classifier in disease prediction.

In addition, the Naive Bayes classifier has been applied in other fields, such as predicting student graduation [11] and detecting network attacks [12]. This study shows the versatility of the Naive Bayes classifier in a variety of prediction tasks. In conclusion, Naive Bayes classifier is a widely used method for prediction tasks in various domains. Its effectiveness has been demonstrated in software error prediction, sentiment analysis, disease prediction, and other fields.

Naive Bayes method is an attractive choice because of its simplicity. Although the assumption that the attributes are independent ("naive" assumption) may not always hold in real contexts, this method often provides good results in classification problems, including software defect prediction. Muzaki and Witanti [9] argue that Naive Bayes (NB) is a good for classification. NB uses probability theory as its theoretical basis and has a high level of speed and accuracy when applied to large databases. NB can determine the class of data during classification by testing all labels on the data using Bayes' theorem. The class that has the highest probability value becomes the prediction from the method.

Python programming language was chosen as a platform for implementing the Naive Bayes method in this research because Python provides a variety of powerful libraries for data analysis and machine learning. A relevant paper reference in the context of using Python for data analysis and machine learning is "Python for Data Analysis" by Wes McKinney [13]. Python also has effective visualization tools such as Matplotlib and Seaborn to visualize data analysis results. Python implementation involves steps such as reading the dataset, processing the data, training the model, and evaluating the performance of the model. Data analysis is an important stage in this research, and the Python method makes it easy to read data, clean it, delete irrelevant data, and explore data to identify patterns or relationships between software attributes and the presence of defects [13]. Evaluation of the performance of device defect prediction models' software is the final stage in this research [14], which is suitable for measuring classification tasks. Performance metrics such as accuracy, precision, and recall will be used to evaluate the model.

The evaluation results support understanding the extent to which Naive Bayes models are effective in predicting software defects. Supervision-based machine learning is used to evaluate machine learning capabilities in

Device Behavior Prediction (SBP). This study discusses Naïve Bayes (NB) classifiers. The discussed machine learning classifiers were applied to three different datasets obtained from the book [15], Previous researchers have developed and applied various bug prediction approaches that differ in terms of accuracy, complexity, and input data that is needed but has not achieved the desired results [16].

Software defect prediction is important because it allows software developers to allocate available resources to create high-quality software products that can help in every company's business processes [17]. By predicting module defects early, developers can identify potential problems early and allocate resources appropriately. The use of Naive Bayes classification in software defect prediction has been proven to be effective. In a study by Hardoni [1] the integration of SMOTE (Synthetic Minority Oversampling Technique) with Naive Bayes and Logistic Regression, based on particle swarm optimization, improved software defect prediction. This study shows that this approach successfully outperforms previous methods in terms of performance.

2. Related Research

This research includes several studies that focus on software defect prediction, input methods used, results, intermediate representations, and limitations of each study. Kaur [18] investigated the prediction of aging-related software bugs (ARBs) using data from a bug repository. They developed the SEARCH_KEYWORD algorithm for ARB prediction with intermediate representation to ARB prediction. However, this study has limitations such as an unbalanced proportion of ARB-prone and ARB-free files, limited availability of training data, and limited comparative analysis to a specific set of classifiers and datasets.

Yalamanchili [19] deals with software defect prediction using machine learning based on historical software defect data. They use supervised machine learning algorithms (Naïve Bayes, SVM, ANN) to predict future software errors. Despite mentioning high accuracy rates, the study lacks specific metrics, does not compare with other machine learning algorithms, and does not address hidden errors or provide detailed information about software adaptation or resource utilization improvements. Yalamanchili focuses on developing a bug severity prediction model using word2vec on text descriptions of bug reports. They use word2vec to embed and predict bug severity with real-valued vectors as intermediate representations [19]. However, this study has limitations, such as time-consuming hyperparameter tuning, performance dependence on hyperparameter configuration, focus on text-based bug reports and varying performance of classifiers based on data and word occurrences.

Wang [20] discusses deep semantic feature learning for software defect prediction using Deep Belief Network. They extract semantic features from token vectors derived from Abstract Syntax Trees (AST) and source code changes. Unfortunately, the abstract lacks specific details about the model and data, which would likely be provided in the full paper.

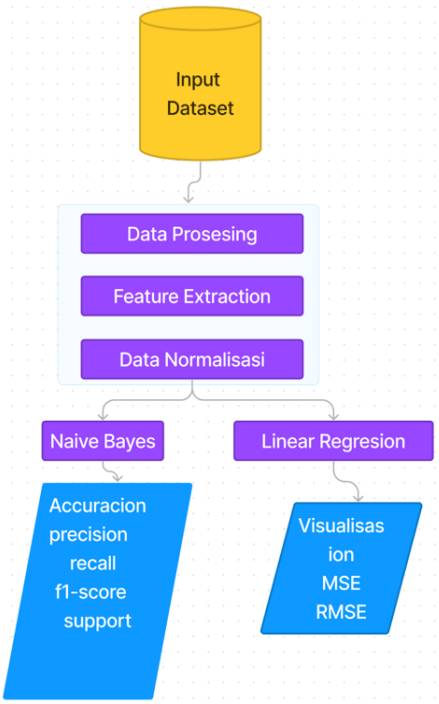
Turhan and Bener [21] analyzes Naive Bayes assumptions on software error data, using publicly available software defect data from NASA. They preprocessed the data using Principal Component Analysis (PCA) and applied various methods to analyze Naive Bayes assumptions. However, this research is limited to an analysis of Naive Bayes assumptions on software defect data from NASA, and the results may not be generalizable to other data sets or domains.

In this study, an implementation of the Naive Bayes method in Python in the Google Colab environment will be used to analyze bug predictions in the JM1 dataset. According to Zaidi [22], attribute weighting in Naive Bayes can help reduce the impact of prediction failure. It is suggested that the Naive Bayes algorithm is

effective for software defect prediction. In a study by Gata [23], the Naive Bayes algorithm achieved an accuracy of 69.18% and an AUC value of 0.771, indicating its classification performance. This implementation aims to leverage the power of Naive Bayes in software defect prediction, specifically in the context of the JM1 dataset, to improve accuracy and predictive capabilities.

Moreover, Gata [23] compared different methods for dealing with imbalanced data in software fault prediction and found that the Naive Bayes classifier assumes conditional independence of attributes, which is suitable for software defect prediction. This highlights the relevance of Naive Bayes classification in this context. It is hoped that the results of this research can become a basis for software developers to improve software quality and identify potential problems early. With more efficient software defect prediction, it can be hoped that the risk and impact of software defects can be minimized.

Figure 1
Research design phase



3. Method

This stage is the core of this research and yields valuable insights into software defect prediction. This research provides a comprehensive view of the implementation from data processing to model testing, as shown in Figure 1. Each stage in this methodology has an important role in achieving the research objectives, namely analyzing software defect predictions using the Naive Bayes and Linear Regression methods and measuring the accuracy of the prediction results.

Data Collection

The data used comes from the JM1-Dataset-Attributes-Prediction dataset, which involves software attributes and defect labels. This dataset provides information regarding software characteristics, and defect labels provide information about whether a software entity has a defect or not. By using this dataset, software attributes can be analyzed to understand patterns and relationships that might influence the possibility of defects in the software. 4301 entries, 0 to 4300 Data columns (total 22 columns), as shown in Table 1.

Table 1
Dataset JM1

loc	v(g)	ev(g)	iv(g)	n	…
11.0	2.0	1.0	2.0	20.0	….
14.0	2.0	1.0	1.0	21.0	…
10.0	2.0	1.0	2.0	15.0	….
5.0	1.0	1.0	1.0	11.0	
10.0	3.0	3.0	1.0	21.0	….
..	….	…	….	….	

Data Exploration and Visualization

Before applying Naive Bayes classification, an understanding of the data distribution needs to be gained. Histograms are used to identify the distribution pattern of each feature in the JM1 dataset, enabling the determination of the presence of outliers or uneven distribution that may impact model performance. To find out the extent to which each feature correlates with each other in the JM1 dataset, covariance is used. The

directional relationship between two variables can be analyzed through covariance. When negative covariance values are encountered, it indicates an inverse relationship, while positive covariance values indicate a tendency for the features to move together.

The effectiveness of visualization of the correlation matrix between features in the dataset is improved using heatmaps. With a heatmap, the relationship between features can be seen clearly, giving rise to correlation patterns that play a role in the feature selection process or a deeper understanding of the data. The advantages of the scatter plot method in seeing the direct relationship between two variables are explored in the JM1 dataset. Scatter plots help identify patterns of relationships between certain pairs of features, facilitating the determination of linear or non-linear patterns that may influence prediction results using Naive Bayes models.

Data Processing

Preliminary data, for example, loc = [11.0, 14.0, 10.0, 5.0, 10.0] to be normalized using Min-Max Scaling in three steps.

1) Calculate the minimum value (min_loc) and maximum value (max_loc) from "loc" feature. min_loc = 5.0 (minimum value) and max_loc = 14.0 (maximum value).
2) Normalize each value in the "loc" feature using formula 1.

$$normalize_{loc} = \frac{loc - \min\_loc}{\max\_loc - \min\_loc} \quad (1)$$

3) Calculate normalized values for each data in the feature "loc".

   normalized_loc = [(11.0 - 5.0) / (14.0 - 5.0), (14.0 - 5.0) / (14.0 - 5.0), (10.0 - 5.0) / (14.0 - 5.0), (5.0 - 5.0) / (14.0 - 5.0), (10.0 - 5.0) / (14.0 - 5.0)]

   so, normalized_loc = [0.6, 1.0, 0.5, 0.0, 0.5]

Feature Extraction

Feature extraction from the Software Defect Prediction Dataset involves an in-depth understanding and analysis of existing attributes [1]. This dataset is provided to support the development of software prediction models that can be repeated, verified, and improved. In the context of analysis, feature extraction is an important initial stage in understanding the characteristics of the software to be analyzed.

Several main attributes in the dataset and how to extract features from each attribute.

- *loc* (McCabe's Line Count of Code): This attribute measures the number of lines of code in the software. In feature extraction, it can be used to measure code complexity by identifying the number of lines involved in the software.
- *v(g)* (McCabe "Cyclomatic Complexity"): This attribute measures software complexity based on the McCabe metric. In feature extraction, this complexity can be used as an indicator for the level of difficulty in understanding and managing the software.
- *ev(g)* (McCabe "Essential Complexity"): Essential complexity is a fixed level of complexity in software. Feature extraction from these attributes can help in understanding the core complexity of the software.
- *iv(g)* (McCabe "Design Complexity"): This attribute measures the complexity of software design. Feature extraction can help in analyzing the extent to which design influences software complexity.
- *n* (Halstead Total Operators + Operands): This attribute includes the total number of operators and operands in the software. Feature extraction from these attributes can be used to measure the number of entities involved in the software.
- *v* (Halstead "Volume"): Halstead volume measuring software size. In feature extraction, these attributes can be used to understand the size of the software being analyzed.

- (Halstead "Program Length"): Program length is a measure of software based on the Halstead metric. Feature extraction from these attributes can help in identifying code length.
- *d* (Halstead "Difficulty"): This attribute measures the level of difficulty in understanding the software. Feature extraction from these attributes can provide insight into the level of complexity of the software.
- *i* (Halstead "Intelligence"): Intelligence in Halstead's context refers to the level of "intelligence" of the code. Feature extraction can help in analyzing the extent to which the code is considered intelligent.
- *e* (Halstead "Effort"): This attribute measures the effort required to develop the software. Feature extraction from these attributes can provide an idea of how much effort is required in development.

Data Normalization (Min-Max Normalization)

The calculation of the Min-Max normalization process can be illustrated, for example, using initial data *v*: [3, 6, 9, 12], b: [15, 18, 21, 24].

Min-Max Normalization for Column *v*

$$v_{scaled} = \frac{v - \min(v)}{\max(v) - \min(v)} \quad (2)$$

$$v_1 = \frac{3 - \min(3)}{\max(12) - \min(3)} = \frac{0}{9} = 0$$

$$v_2 = \frac{6 - \min(3)}{\max(12) - \min(3)} = \frac{3}{9} = \frac{1}{3}$$

$$v_3 = \frac{9 - \min(3)}{\max(12) - \min(3)} = \frac{6}{9} = \frac{2}{3}$$

$$v_1 = \frac{12 - \min(3)}{\max(12) - \min(3)} = \frac{9}{9} = 1$$

So, the Min-Max normalization results for the column $v_{scaled} = \left[0, \frac{1}{3}, \frac{2}{3}, 1\right]$.

Then, normalization Min-Max for *b*

$$b_{scaled} = \frac{b - \min(b)}{\max(b) - \min(b)} \quad (3)$$

$$b_1 = \frac{15 - \min(15)}{\max(24) - \min(15)} = \frac{0}{9} = 0$$

$$b_2 = \frac{18 - \min(15)}{\max(24) - \min(15)} = \frac{3}{9} = \frac{1}{3}$$

$$b_3 = \frac{21 - \min(15)}{\max(24) - \min(15)} = \frac{6}{9} = \frac{2}{3}$$

$$b_4 = \frac{24 - \min(15)}{\max(24) - \min(15)} = \frac{9}{9} = 1$$

So, the Min-Max normalization results for $b_{scaled} = \left[0, \frac{1}{3}, \frac{2}{3}, 1\right]$.

Prediction Model Implementation

Next, the training process is carried out using previously normalized data, including features *v* and *b*, as well as software defect classification targets which are determined as 0 for no defect and 1 for defect.

Naive Bayes model uses the likelihood and posterior functions as a formula 4.

$$P(defect|Data) = \frac{P(Data|Defect) * P(Defect)}{P(Data)} \quad (4)$$

To predict the software defect rate, a Linear Regression model has been applied. The model uses additional data on actual defect rates and features *v* and *b*. Linear Regression for defect prediction uses formula 5.

$$Defect = \beta 0 + \beta 1 * v_{scaled} + \beta 2 * b_{scaled} \quad (5)$$

Coefficients β0, β1, and β2 in the Linear Regression model have been determined through a training process using normalized data. This model can be used to predict defect rates based on new data provided.

Model Testing

Suppose the predicted result is $\hat{y}$ = [8, 12, 18, 22] from the model for the software defect rate and the actual value y = [10, 15, 20, 25].

MSE is calculated by adding the squared differences between each prediction and the actual value and then dividing by the number of observations.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \quad (6)$$

For the above example, *obtain the value of MSE*= $41((8-10)^2+(12-15)^2+(18-20)^2+(22-25)^2)$

MSE=41(4+9+4+9) = 6.5

Another measurement is RMSE, the square root of MSE, giving an idea of how big the average error is.

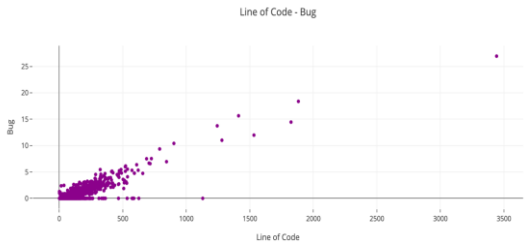
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \qquad (7)$$

RMSE=6.5≈2.55

4. Result and Analysis

The results of classification with Naïve Bayes can be visualized with a plot, as shown in Figure 2.

Figure 2
Plot visualization of LoCK Bug Classification results



Linear regression results in predicting the level of software defects, which produces Intercept and Coef values. The Intercept value is -0.09359968. This is the value that represents the point where the regression line intersects the vertical axis (y-axis) when all independent variables (x) are zero. In the context of this model, the Intercept value indicates the starting point or baseline of the prediction. This means that if all independent variables are zero, then the predicted value will be close to -0.09359968. This is the baseline value before the influence of other independent variables is considered.

The Coef value is [0.00761893]. This is the coefficient that connects the independent variable (x) with the dependent variable (y) in the regression model. In this context, the coefficient indicates how much influence a one-unit increase in the independent variable will contribute to the prediction of the dependent variable. With a coefficient value of 0.00761893, every one-unit increase in the independent variable will cause an increase of approximately 0.00761893 units in the prediction.

So, these results explain how this regression model calculates predictions based on a combination of Intercept and Coef values as well as the values of the existing independent variables. The Intercept value is the starting point, and the Coef is a coefficient that shows the extent to which each independent variable influences the final prediction.

5. Discussion

This experiment aims to test various combinations of the percentage of "True" and "False" values in the training data and observe their effect on the accuracy of the model used. This experiment uses the Naive Bayes (NB) algorithm as a predictive model.

Table 2
Prediction results and accuracy

Percentage		Accuracy NB	Accuracy	
False	True		RMSE	MSE
10%	90%	0.954	0.478	0.228
25%	75%	0.967	0.372	0.138
40%	60%	0.975	0.383	0.147
75%	25%	0.980	0.395	0.198

It can be observed that the composition of the training data has a significant influence on accuracy, with some combinations producing better accuracy than others. The results of this experiment can be used as a guide in selecting the most appropriate percentages for training data in the Naive Bayes model.

In the literature review, it was found that when evaluating classification algorithms in software defect prediction, previous studies tend to use metrics such as accuracy, recall, and F1 score. All three are based on the confusion matrix shown in Table 3 and 4.

Table 3
Accuracy descriptive statistics for the JM1 data set.

Algorithm	Median	Standard deviation
K2	0.8062	0.45
Hill Climbing	0.8062	0.45

TAN	0.8069	0.76
Naive Bayes	0.98	0.007367

Table 4
Accuracy results on the same dataset JM1

Algorithm	Best Accuracy
K2	0.8079
Hill Climbing	0.8079
TAN	0.8236
Decision Tree	0.8170
Naive Bayes model	0.98

6. Conclusion

Based on the results of testing the entire dataset, it can be concluded that the Naïve Bayes method produces a classification with an accuracy of 0.98. For the results of the linear regression, it was found that this model had an Intercept value of -0.09359968647139849 and a Coef coefficient of 0.00761893. The "Intercept" value represents the starting point or baseline of the prediction in the context of this model, while the Coef coefficient shows how much influence changes in the independent variable have on the prediction of the dependent variable.

References

[1] A. Hardoni, "Integrasi SMOTE pada Naive Bayes dan Logistic Regression Berbasis Particle Swarm Optimization untuk Prediksi Cacat Perangkat Lunak," *J. Sist. dan Teknol. Inf.*, vol. 9, no. 2, p. 144, Apr. 2021, doi: 10.26418/justin.v9i2.43173.

[2] E. Dantas, A. Sousa Neto, M. Perkusich, H. Almeida, and A. Perkusich, "Using Bayesian Networks to Support Managing Technological Risk on Software Projects," in *Anais do I Workshop Brasileiro de Engenharia de Software Inteligente (ISE 2021)*, Sociedade Brasileira de Computação, Sep. 2021, pp. 1–6. doi: 10.5753/ise.2021.17277.

[3] I. Ancveire, I. Gailite, M. Gailite, and J. Grabis, "Software Delivery Risk Management: Application of Bayesian Networks in Agile Software Development," *Inf. Technol. Manag. Sci.*, vol. 18, no. 1, Jan. 2015, doi: 10.1515/itms-2015-0010.

[4] S. Das, A. Mudgal, A. Dutta, and S. R. Geedipally, "Vehicle Consumer Complaint Reports Involving Severe Incidents: Mining Large Contingency Tables," *Transp. Res. Rec. J. Transp. Res. Board*, vol. 2672, no. 32, pp. 72–82, Dec. 2018, doi: 10.1177/0361198118788464.

[5] B. Assemi, M. Hickman, and A. Paz, "Relationship between Programmed Heavy Vehicle Inspections and Traffic Safety," *Transp. Res. Rec. J. Transp. Res. Board*, vol. 2675, no. 10, pp. 1420–1430, Oct. 2021, doi: 10.1177/03611981211016458.

[6] S. A. Putri, "Prediksi Cacat Software Dengan Teknik Sampel Dan Seleksi Fitur Pada Bayesian Network," *J. Kaji. Ilm.*, vol. 19, no. 1, p. 17, Jan. 2019, doi: 10.31599/jki.v19i1.314.

[7] N. Ichsan, H. Fatah, E. Ermawati, I. Indriyanti, and T. Wahyuni, "Integrasi Distribution Based Balance dan Teknik Ensemble Bagging Naive Bayes Untuk Prediksi Cacat Software," *Media J. Inform.*, vol. 14, no. 2, p. 79, Dec. 2022, doi: 10.35194/mji.v14i2.2623.

[8] N. Hidayati, J. Suntoro, and G. G. Setiaji, "Perbandingan Algoritma Klasifikasi untuk Prediksi Cacat Software dengan Pendekatan CRISP-DM," *J. Sains dan Inform.*, vol. 7, no. 2, pp. 117–126, Nov. 2021, doi: 10.34128/jsi.v7i2.313.

[9] A. Muzaki and A. Witanti, "Sentiment Analysis of The Community in The Twitter to The 2020 Election in Pandemic Covid-19 By Method Naive Bayes Classifier," *J. Tek. Inform.*, vol. 2, no. 2, pp. 101–107, Mar. 2021, doi: 10.20884/1.jutif.2021.2.2.51.

[10] R. Yuliza, "Sistem Pakar Akurasi dalam Mengidentifikasi Penyakit Gingivitis pada Gigi Manusia dengan Metode Naive Bayes," *J. Sistim Inf. dan Teknol.*, Aug. 2022, doi: 10.37034/jsisfotek.v5i1.157.

[11] K. R. Diska and K. Budayawan, "Sistem Informasi Prediksi Kelulusan Menggunakan Metode Naive Bayes Classifer (Studi Kasus: Prodi Pendidikan Teknik Informatika)," *J. Pendidik. Tambusai*, vol. 7, no. 1, pp. 936–943, Feb. 2023, doi: 10.31004/jptam.v7i1.5375.

[12] D. K. Nurilahi, R. Munadi, S. Syahrial, and A. BAHRI, "Penerapan Metode Naïve Bayes pada Honeypot Dionaea dalam Mendeteksi Serangan Port Scanning," *ELKOMIKA J. Tek. Energi Elektr. Tek. Telekomun. Tek. Elektron.*, vol. 10, no. 2, p. 309, Apr. 2022, doi: 10.26760/elkomika.v10i2.309.

[13] W. McKinney, *Python for data analysis: Data wrangling with pandas, NumPy, and IPython*, Second. O'Reilly Media, Inc, 2022.

[14] Warto *et al.*, "Systematic Literature Review on Named Entity Recognition: Approach, Method, and Application," *Stat. Optim. Inf. Comput.*, vol. 12, no. 4, pp. 907–942, Feb. 2024, doi: 10.19139/soic-2310-5070-1631.

[15] Y. Tohma, K. Tokunaga, S. Nagase, and Y. Murata, "Structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution," *IEEE Trans. Softw. Eng.*, vol. 15, no. 3, pp. 345–355, Mar. 1989, doi: 10.1109/32.21762.

[16] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, IEEE, May 2010, pp. 31–41. doi: 10.1109/MSR.2010.5463279.

[17] N. P. Gargote, S. Devaraj, and S. Shahapure, "Human Perception Based Color Image Segmentation," *Comput. Eng. Appl. J.*, vol. 2, no. 3, pp. 283–294, 2013, doi: 10.18495/comengapp.v2i3.34.

[18] H. Kaur and A. Kaur, "An empirical study of Aging Related Bug prediction using Cross Project in Cloud Oriented Software," *Informatica*, vol. 46, no. 8, Nov. 2022, doi: 10.31449/inf.v46i8.4197.

[19] P. L. S. T. Sangeetha Yalamanchili, "Software Defect Prediction Using Machine Learning," *Int. J. Recent Technol. Eng.*, vol. 8, no. 2S11, pp. 1053–1057, Nov. 2019, doi: 10.35940/ijrte.B1178.0982S1119.

[20] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep Semantic Feature Learning for Software Defect Prediction," *IEEE Trans. Softw. Eng.*, vol. 46, no. 12, pp. 1267–1293, Dec. 2020, doi: 10.1109/TSE.2018.2877612.

[21] B. Turhan and A. Bener, "Analysis of Naive Bayes' assumptions on software fault data: An empirical study," *Data Knowl. Eng.*, vol. 68, no. 2, pp. 278–290, Feb. 2009, doi: 10.1016/j.datak.2008.10.005.

[22] N. A. Zaidi, J. Cerquides, M. J. Carman, and G. I. Webb, "Alleviating Naive Bayes Attribute Independence Assumption by Attribute Weighting," *J. Mach. Learn. Res.*, vol. 14, no. 24, pp. 1947–1988, 2013.

[23] W. Gata *et al.*, "Algorithm Implementations Naïve Bayes, Random Forest. C4.5 on Online Gaming for Learning Achievement Predictions," in *Proceedings of the 2nd International Conference on Research of Educational Administration and Management (ICREAM 2018)*, Paris, France: Atlantis Press, 2019. doi: 10.2991/icream-18.2019.1.

This page is intentionally left blank.