INFOTECH
Journal of Applied Sciences and Knowledge Society
Volume 2, No. 1, January/December 2021

Predicting Software Defects Using Bayesian Approaches

Samuel King [1]

[1] University of Sydney - Australia,
kingsamuel081983@gmail.com,https://orcid.org/0009-0002-8194-7943

ABSTRACT

In the realm of software engineering, the anticipation of software flaws holds significant importance as it enables developers to pinpoint and rectify issues before they escalate into expensive and challenging bugs. Timely identification of software defects not only economizes time and resources in the software development lifecycle but also assures the ultimate quality of the end product. This study seeks to assess three algorithms for constructing Bayesian Networks, aiming to classify projects as susceptible to defects. While Naive Bayes is the prevailing method in literature, this research introduces K2, Hill Climbing, and TAN as alternatives for constructing Bayesian Networks. Meanwhile, three publicly available PROMISE datasets are employed, incorporating McCabe and Halstead complexity metrics. The obtained results are benchmarked against widely used approaches like Decision Tree and Random Forest. Performance metrics applied in a cross-validation process reveal that the classification outcomes are on par with Decision Tree and Random Forest. Notably, Bayesian algorithms exhibit lower variability, enhancing the robustness of software engineering predictions. This advantage is evident in the consistent results of training and test data selection, distinguishing them from the variable outcomes observed in Decision Tree and Random Forest approaches.

Keywords: *defect anticipation in software; Bayesian Networks; categorization; machine learning*

Received:
09 August, 2021

Accepted for publication:
22 September, 2021

INFOTECH
Journal of Applied Sciences and Knowledge Society
Volume 2, No. 1, January/December 2021

Predicción de Defectos de Software Utilizando Enfoques Bayesianos

RESUMEN

En el ámbito de la ingeniería de software, la anticipación de fallas en el software tiene una importancia significativa, ya que permite a los desarrolladores identificar y corregir problemas antes de que se conviertan en errores costosos y desafiantes. La identificación oportuna de defectos de software no solo ahorra tiempo y recursos en el ciclo de desarrollo de software, sino que también garantiza la calidad final del producto. Este estudio busca evaluar tres algoritmos para construir Redes Bayesianas, con el objetivo de clasificar proyectos como susceptibles a defectos. Mientras que Naive Bayes es el método predominante en la literatura, esta investigación presenta K2, Hill Climbing y TAN como alternativas para construir Redes Bayesianas. Mientras tanto, se emplean tres conjuntos de datos públicos de PROMISE, que incorporan métricas de complejidad de McCabe y Halstead. Los resultados obtenidos se comparan con enfoques ampliamente utilizados como Decision Tree y Random Forest. Las métricas de rendimiento aplicadas en un proceso de validación cruzada revelan que los resultados de clasificación están a la par con Decision Tree y Random Forest. Es importante destacar que los algoritmos bayesianos muestran una menor variabilidad, mejorando la robustez de las predicciones en ingeniería de software. Esta ventaja es evidente en los resultados consistentes de la selección de datos de entrenamiento y prueba, lo que los distingue de los resultados variables observados en los enfoques de Decision Tree y Random Forest.

Palabras clave: *anticipación de defectos en software; Redes Bayesianas; categorización; aprendizaje automático*

INTRODUCTION

The presence of software defects poses significant challenges and inconveniences in both software development and maintenance, negatively impacting the overall quality of the software. Despite meticulous processes during development, achieving completely defect-free software remains elusive. Consequently, software testing becomes a pivotal phase in the software development life cycle, serving as a means to proactively prevent or rectify potential software failures before deployment. However, the testing process is often intricate, demanding thorough planning and substantial resources [1]. Software defects exert substantial influence on performance, quality, costs, and user satisfaction. Consequences of a high number of defects include delivery delays, unforeseen costs, subpar user experiences, diminished customer confidence, and even security vulnerabilities. These repercussions directly compromise software quality.

Given the adverse effects of detecting defects late in the development stages, the field of software defect prediction (SDP) emerges, involving the creation of prediction models based on historical data to anticipate future software faults [2]. Predicting defects becomes essential for identifying potentially flawed modules in the software, facilitating the development of an efficient, reliable, and cost-effective software product in a timely manner. The identification of defect-prone modules enables the allocation of resources to prevent unforeseen costs.

Developing a software defect prediction model is a challenging task, and artificial intelligence, particularly machine learning (ML) algorithms, can play a supportive role in predicting defects early in the development process. Research by Hammouri et al. [2] discusses classifiers such as Naïve Bayes (NB), Decision Trees (DT), and Artificial Neural Networks (ANN) for defect prediction. However, Bayesian approaches, known for solving diverse problems across various domains, are considered as alternatives that may offer superior precision [3].

Additionally, research by Herzig et al. [4] reveals that a considerable percentage of problem reports in bug databases were misclassified, impacting the accuracy of defect prediction. Incorrect data quality undermines the achievement of high prediction accuracy. Consequently, there is a need to explore the performance and precision of lesser-explored algorithms based on Bayesian Networks. These

algorithms aim to provide software engineers with greater certainty in making estimates and delivering high-quality products.

In a comprehensive examination of the literature conducted in [5], 38 studies from the period between 2016 and 2020 focused on software defect prediction, aiming to scrutinize the prevalent classification approaches and algorithms in this domain. The analysis revealed that the most widely employed approaches involve ensemble algorithms, with Random Forest [6,7] taking the lead, followed by other algorithms like AdaBoost [8] and Bagging [9]. Similarly, approaches grounded in Bayes' theorem were identified, although they exclusively relied on the Naive Bayes algorithm and its variants [7,10,11].

Alternative methodologies, such as Decision Trees, specifically highlighting the prevalence of the C4.5 algorithm, were also noted as extensively reported [10]. Furthermore, simpler classifiers, including Support Vector Machine [12–14], K-Nearest-Neighbor [14], and Logistic Regression [13,15], were utilized in various studies.

Recent works have undertaken comparisons between different approaches [16–18], particularly focusing on ensemble methods that employ multiple algorithms and contrasting them with classical approaches. Some studies, such as [19], delve into experimentation with various versions of a specific algorithm, such as Support Vector Machine (SVM).

Motivation

The impetus behind this research is to aid software engineers in constructing defect prediction models that are both more accurate and robust—models whose outcomes are not overly influenced by variations in training data. This becomes crucial as effective defect prediction empowers teams to pinpoint the areas and modules within the software most susceptible to bugs or defects. Such insights enable development and testing teams to concentrate their efforts on these critical areas, thereby mitigating the risk of defects spreading and compromising the overall functionality and quality of the software.

Problem Statement

A notable observation from the studies presented in Table 1 is that most of them only report the best result. This aspect is significant because machine learning algorithms are often sensitive to training data, exhibiting varying performance against test data. Achieving better results is contingent on the similarity between test and training data. Hence, it is imperative to report the performance of classification algorithms, ensuring not only precision but also robustness across diverse datasets. Commonly employed cross-validation procedures are instrumental in generating models with distinct training data subsets from the dataset and assessing their performance against different test subsets that were not part of the training process.

Table 1 presents a synopsis and comparative analysis between our research and recent studies (NR = Not Reported, DT = Decision Tree, EM = Ensemble Methods, NB = Naïve Bayes, NN = Neural Networks, KNN = K-Nearest Neighbor, SVM = Support Vector Machine, LR = Logistic Regression, and BN = Bayesian Network).

Ref	Year	Approaches Used	Bayesian Networks Use	Performance Analysis	Comparison with Other Similar Approaches or Variants	Comparison with Other Approaches
[6]	2019	DT, EM	NO	NO	YES	YES
[7]	2020	DT, EM	NO	NO	NO	YES
[9]	2020	EM	NO	YES	YES	NO
[10]	2018	NB, DT, EM, SVM	NO	NO	NO	YES
[11]	2020	NB, DT, NN	NO	YES	NO	YES
[12]	2020	DT, NB, LR, EM, SVM	NO	NO	NO	YES
[14]	2019	SVM	NO	YES	YES	NO
[15]	2018	EM, NN, DT	NO	NO	NO	YES
[16]	2022	LR, SVM, KNN, DT	NO	NO	NO	YES
[17]	2022	SVM, DT, KNN, NN, NB	NO	NO	NO	YES
[18]	2022	LR, DT, EM	NO	NO	NO	YES
[19]	2022	SVM	NO	YES	YES	NO
Our Research	2023	BN, EM, DT	YES	YES	YES	YES

Another notable observation from Table 1 is the existence of various versions of the same algorithm, with the general version typically being the one reported. Additionally, when assessing an algorithm within a specific approach, such as Decision Tree, Ensemble Methods, or KNN, among others, many studies do not include comparisons with alternative approaches. Notably, no works cited in the literature employed Bayesian Networks, although the conventional Naive Bayes algorithm was commonly utilized. We speculate that the limited adoption of Bayesian Networks in software engineering may stem from a potential lack of familiarity with this particular approach. However, we believe it holds numerous advantages, as elaborated upon in this article, making it a promising option for enhancing performance.

Contribution

The primary contribution of our research is outlined below

- This paper empirically presents the outcomes of employing classification algorithms based on Bayes' theorem, a facet not extensively reported in the existing literature on software defect prediction. Specifically, various methods of constructing Bayesian Networks are explored to offer software engineers alternative strategies for predicting defects in their projects.
- The selection of Bayesian Networks is justified by the recognition that merely knowing the value of the class variable is insufficient for software engineers and testers. It is equally crucial to understand the characteristics that warrant heightened attention. Additionally, certain algorithms, like KNN or Random Forest, may not explicitly express the variables most influential in the classification task.
- The tests are conducted on the well-established public PROMISE repository, ensuring repeatability and comparability of results.
- A statistical comparison is conducted on the tests using a 10-fold cross-validation method, a common practice in the literature. This analysis unveils the performance of the algorithms across different training and test data, providing insights into both optimal and suboptimal outcomes and assessing the variability in their performance.
- The proposed methods in the experimentation are compared with two approaches mentioned in the literature, namely J48 (Decision Trees approach) and Random Forest (ensemble algorithms approach).
- Notably, the comparison in this paper refrains from including the precision metric due to its bias towards the class value that appears most frequently in unbalanced datasets. Instead, metrics such as recall, accuracy, and F1-measure are utilized, capturing diverse aspects of a model's performance.
- The results are deliberated upon to strike a balance between precision and robustness in the various methods tested.

- The significance of this work lies in the realm of software defect prediction, offering developers and project leaders insights into when a system can be released, thereby reducing the use of unforeseen resources and enhancing the user experience by minimizing the number of defects.

Structure of the Paper

The organization of this document is as follows: Section 2 provides an overview of related works that informed the decisions made for the experimentation in this research. Section 3 outlines our proposed approach. Section 4 details the characteristics of the utilized datasets. Section 5 delves into the experiments, evaluation criteria, and analysis of the primary results. Lastly, Section 6 concludes the paper and outlines directions for future work.

Related Work

The literature cited in Section 1 played a pivotal role in shaping the focus of this research. While the Bayesian approach is widely employed, our exploration of Bayesian Networks, distinct from the commonly used Naive Bayes, presents a valuable opportunity. Moreover, the methodologies introduced in this study will be juxtaposed with the algorithms and approaches most prevalent in the existing literature. Specifically, an ensemble approach like Random Forest and a Decision Tree, exemplified by C4.5, are chosen for comparison.

Table 2 displays the metrics utilized to assess the performance of classifiers and presents their corresponding results. Key metrics highlighted include precision, recall, F1-measure, accuracy, and area under the curve. This selection is significant because the optimal evaluation of an algorithm depends on the metric employed. Therefore, it is crucial to assess algorithms using various metrics, as they capture different aspects of performance and can address issues such as class imbalance or overfitting. The most widely recognized metrics were chosen to ensure a comprehensive evaluation of the research results. However, it is worth noting that, as indicated in Table 2, several related works concentrate solely on describing the precision of a proposal using a single metric. This raises suspicion, considering the nature of the data necessitates the use of different metrics based on the class, rather than relying solely on percentage accuracy.

Table 2 provides a summary of metrics and validation methods documented in recent research, denoting instances where information is not reported with "NR" for clarity.

Ref	Data Set	Cross Validation	Best Approach	Accuracy	Recall	F1-Measure
[6]	CM1	10-fold	Random Forest/Sampling	NR	NR	0.92
[6]	JM1	10-fold	Random Forest/Sampling	NR	NR	0.85
[6]	KC1	10-fold	Random Forest/Sampling	NR	NR	0.87
[7]	CM1	10-fold	Random Forest	0.97	0.97	0.97
[7]	JM1	10-fold	Random Forest	0.81	0.81	0.77
[7]	KC1	10-fold	Random Forest	0.83	0.85	0.83
[8]	CM1	NR	Ada Boost	0.87	NR	NR
[8]	JM1	NR	Ada Boost	0.89	NR	NR
[8]	KC1	NR	Ada Boost	0.85	NR	NR
[14]	CM1	10-fold	Support Vector Machine	0.88	0.89	0.82
[14]	JM1	10-fold	Support Vector Machine	0.81	0.82	0.79
[14]	KC1	10-fold	Support Vector Machine	0.84	0.83	0.84
[16]	CM1	NR	Random Forest	0.86	NR	NR
[16]	JM1	NR	Random Forest	0.34	NR	NR
[16]	KC1	NR	Naive Bayes	0.77	NR	NR
[17]	CM1	NR	K-NN undersample	0.89	NR	NR
[17]	JM1	NR	SVM undersample	0.93	NR	NR
[17]	KC1	NR	K-NN undersample	0.96	NR	NR
[19]	CM1	NR	SVM-Linear	0.79	NR	NR
[19]	JM1	NR	SVM-RBF	0.88	NR	NR
[19]	KC1	NR	SVM-Linear	0.83	NR	NR

Moreover, Table 2 reveals that a considerable number of studies do not specify a model validation method. This omission is significant because the selection of data for training and testing the models plays a crucial role in influencing the obtained results.

Our Proposal

While various algorithms exist for predicting software defects with reasonable accuracy, it is imperative to explore alternative classification methods that can achieve higher accuracy rates in software defect prediction. The selected algorithm should positively impact software reliability, quality, and mitigate high costs. Enhancing software defect accuracy implies improvements in developer performance, reduced testing times, and more efficient resource allocation for project managers. We prioritize this approach due to its advantages over other models.

The primary rationale for choosing Bayesian Networks lies in their capability to model causal relationships between variables, aiding in understanding the interplay of features in a software product that influence its susceptibility to defects. Additionally:

1. Bayes' theorem inherently handles uncertainty in the data, which is crucial when dealing with noisy or incomplete data.

2. Bayesian Networks are flexible and can accommodate various types of variables, including continuous and discrete variables, allowing the incorporation of diverse software metrics.
3. Unlike some machine learning algorithms (e.g., K-Nearest Neighbor or Random Forest), the structure of Bayesian Networks is interpretable. This interpretability is essential for software engineers who can visually interpret relationships between software attributes through a graph and focus on those influencing a defective product negatively.

Bayesian Approach

Bayes' Theorem, developed by the British mathematician and theologian Thomas Bayes, is a proposition used to calculate the conditional probability of an event. This theorem aims to determine the probability of one event in comparison to the probability of a similar event. In essence, it enables the calculation of the conditional probability of an event, denoted as A given B, wherein the probability distribution of event B given A is analyzed [20].

The Bayes formula, also known as Bayes' rule, encompasses three distinct probabilities, as shown in Equation (1): P(A) represents the a priori probability of event A, P(A|B) denotes the posteriori probability of event A, and P(B|A) signifies the probability of event B based on the information from event A.

$$P(A|B) = \frac{(P(B|A) * P(A))}{P(B)}$$

Bayesian Networks

The Bayesian Network, a classifier grounded in Bayes' theorem, serves as its fundamental underpinning. This network is a graphical model illustrating variables, often termed nodes, within a dataset and the probabilistic or conditional dependencies existing among them. While a Bayesian Network can reveal causal relationships between nodes, the links in the network, also referred to as edges, may not necessarily indicate direct cause-and-effect connections.

Contrary to this, Bayesian Networks belong to the category of probabilistic models utilizing Bayesian inference for probability calculations. They aim to depict conditional dependency and causality by

representing these dependencies through edges in a directed graph. These relationships facilitate efficient inference about random variables within the graph through the utilization of factors.

This classifier was chosen due to its compact, flexible, and interpretable representation of a joint probability distribution. Furthermore, it proves valuable for knowledge discovery as directed acyclic graphs portray causal relationships between variables [21]. Additionally, this model provides crucial insights into the relationships among variables, which can be interpreted as cause-and-effect relationships. Figure 1 illustrates the structure of a Bayesian Network in the form of a directed acyclic graph, reflecting the relationships among variables representing conditional probabilities. For instance, variable C is conditional on variables A and B.

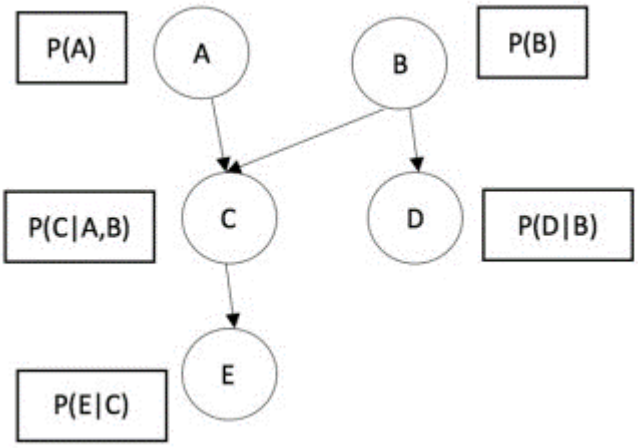

Figure 1. Directed Acyclic Graph (DAG) Representation of a Bayesian Network

Nevertheless, there isn't a singular approach to generating a Bayesian Network, and the method of initialization and construction will depend on the obtained results. To address this, three methods for constructing a Bayesian Network, intended for application in the context of software defect prediction, are outlined below.

TAN

Also recognized as Tree Augmented Naïve Bayesian Network, this algorithm involves constructing a dependency tree connecting the variables to be predicted, with these variables serving as children of the class variable. Consequently, the probability of these variables is calculated by applying Bayes' theorem based on the probability of the class variable [22]. In essence, TAN assumes conditional

independence among all variables given the class variables, thereby permitting predictor variables to depend on each other. Figure 2 illustrates how TAN forms a Bayesian Network, where the variable class lacks parents, and the features (attributes) have the variable class and, at most, one additional attribute as parents.

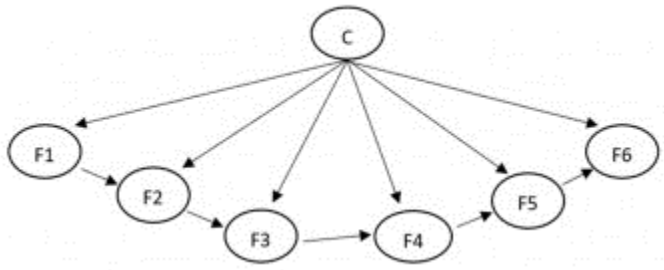

Figure 2. Initialization of Bayesian Network Structure using TAN

Hill Climbing

Hill Climbing, an optimization algorithm, commences with the initiation of a randomly generated Bayesian Network [23]. The algorithm iteratively introduces or removes relationships for each node or feature, calculating the probability of each node within the network based on the joint probability of the class variable. The algorithm selects the optimal network with the highest quality, discarding those that fail to meet its standards. The commonly employed score function for Bayesian Networks is the log-likelihood function, quantifying the probability of the observed data given the network structure and parameters. In essence, this function gauges how effectively the network predicts the data, as illustrated in Equation (2), where S represents the score, G denotes the network structure, D is the data, Xi signifies the i-th variable, and Pai represents the set of parents of Xi in G.

$$S(G,D) = \sum(\log(P(X_i|Pa_i))) - \sum(\log(P(X_i)))$$

The K2 algorithm functions as a heuristic search, initiating with the most straightforward network—one devoid of edges—and presupposes an ordered arrangement of nodes [24]. Embracing the concept of the greedy algorithm, a quintessential structure-learning algorithm [25], K2 streamlines the process of acquiring the Bayesian Network structure, obviating the need for substantial expert knowledge in the problem domain. For each variable in the problem, the algorithm augments its parent set with the

node possessing the lowest probability, resulting in a maximal increase in quality corresponding to the chosen quality measure in the rating process. This iterative process continues until either the quality ceases to increase or a complete Bayesian Network is attained.

Building upon these three approaches for generating a Bayesian Network, the intention is to assess their performance using the mentioned datasets.

Data Sets

The datasets utilized for evaluating the selected algorithms were sourced from the PROMISE repository [26]. The rationale behind selecting these datasets lies in their public availability. PROMISE datasets are made accessible to encourage the development of repeatable, verifiable, refutable, and/or improvable predictive models in software engineering. Furthermore, PROMISE stands out as one of the most widely utilized repositories for predicting software defects, as indicated by results obtained in the RSL. Table 3 presents the chosen datasets, including their instance counts and the distribution of defect classes.

Table 3. Class Distribution Across Datasets

Data Set	Number of Instances	Distributions of Class (Defects)	
		False	True
CM1	498	90.16%	9.83%
JM1	10,885	19.35%	80.65%
KC1	2109	15.45%	84.54%

CM1 represents a NASA spacecraft instrument implemented in the "C" language, JM1 is written in "C" and serves as a real-time predictive ground system, while KC1 is a "C++" system responsible for executing storage management for receiving and processing ground data. The discrete class variable signifies whether the system exhibits defects or is defect-free.

Each dataset comprises 21 explanatory variables or features. Table 4 illustrates the categorization and description of these features.

Table 4. Breakdown of Attribute Types

Type of Attributes	Number of Metrics
Line of Code	5
McCabe measure	3
Base Halstead measure	4
Derived Halstead measure	8
Branch count	1
Total	21

McCabe Metrics

McCabe posited that code featuring intricate pathways is more susceptible to errors. Consequently, his metrics center on capturing the pathways within a code module [27].

Cyclomatic complexity, a key metric, is derived from counting the number of individual logical paths in a program. Thomas McCabe employed graph theory and flow to calculate software complexity. The program is portrayed as a graph, with each instruction serving as a graph node. The potential paths of execution from an instruction (node) are depicted as edges in the graph.

As per the PROMISE repository, McCabe metrics comprise a trio of software metrics, as detailed in Table 5.

Table 5. Metrics According to McCabe's Methodology

ID	Attribute	Value Type
1	V(g): cyclomatic complexity	numeric
2	Ev(g): essential complexity	numeric
3	Iv(g): Design complexity	numeric

Halstead Metrics

Halstead introduced a precise method for gauging a program's size [28]. His approach involves viewing the code as comprised of units termed operators and operands, akin to the tokens discernible by a compiler in that code [29]. Moreover, these operators and operands don't uniformly contribute to complexity. It's imperative to consider not only the total count of elements (operands and operators) but also the count of distinct elements, representing the program's language. Table 6 delineates the fundamental and derived attributes of Halstead metrics.

Table 6. Metrics According to Halstead's Methodology

ID	Measure Type	Attribute	Value Type
1	Base	Unique operators	numeric
2	Base	Unique operands	numeric
3	Base	Total operators	numeric
4	Base	Total operands	numeric
5	Derived	N (operators + operands)	numeric
6	Derived	Volume	numeric
7	Derived	Program length	numeric
8	Derived	Difficulty	numeric
9	Derived	Intelligence	numeric
10	Derived	Effort to write a program	numeric
11	Derived	Time to write a program	numeric
12	Derived	Halstead metric	numeric

Additional Metrics

Among the remaining six metrics, five pertain to lines of code, and one pertains to branch count. Table 7 outlines the attributes of these final metrics, indicating that for the count of code lines, one is proposed by McCabe, and four are proposed by Halstead. The branch count is derived from the flow graph.

Table 7. Additional Metrics

ID	Measure Type	Attribute	Value Type
1	McCabe	Line count of code	numeric
2	Halstead	Line count	numeric
3	Halstead	Comment lines count	numeric
4	Halstead	Blank lines Count	numeric
5	Halstead	Code and Comment	numeric
6	-	Branch count	numeric

Data Preprocessing

Upon acquiring the data, our initial step involves checking for any missing data. Despite the identification of outliers in the datasets, we opted not to remove them. The rationale behind this decision is rooted in the fact that, as we did not gather the data ourselves, we cannot ascertain whether these outliers are the result of data capture errors or genuinely represent atypical data. This underscores a drawback of utilizing publicly available data captured by other individuals, as we lack direct access to the respective projects.

Experiments and Results

This section delineates the methodology employed to validate models in the experimentation, the metrics applied, and the noteworthy results obtained during the assessment of these Bayesian approaches using the PROMISE defect prediction datasets. All experiments were conducted on a platform utilizing Weka 3.9.6, compatible with a Windows 10 Operating system equipped with an Intel Core i7 3.6 GHz processor and 8 GB RAM. The experimental parameters, outlined in Table 8, provide details for reproducibility. It is worth noting that some configuration parameters are specific to particular search algorithms.

Table 8. Configuration of Experimental Parameters in Weka Software

Parameter	Value	Search Algorithm
Batch size	100	All
Score Type	MDL	All
Random order	False	K2
Init as Naïve Bayes	True	K2
Use Arc Reversal	False	Hill Climbing
Markov Blanket Classifier	False	K2 and Hill Climbing

Cross-Validation

Cross-validation, a technique employed in machine learning and statistics, serves to assess a model's performance on data that were not utilized during the training phase. The selection of cross-validation stems from its ability to ensure instances of the faulty class are present in every test set, thereby diminishing the likelihood of classification uncertainty [30]. Instances are partitioned into "folds," and in each iteration, the instances within each fold act as test data, while the remaining instances constitute the training data for model construction. The computed errors are then averaged across all runs.

The choice of the number of folds was influenced by a review of experiments detailed in the literature, as outlined in Section 2. Notably, all these experiments uniformly employed 10-fold cross-validation. Consequently, we opted for the same configuration, utilizing ten folds to evaluate our models consistently.

Model Evaluation Metrics

Upon scrutinizing the literature, it was observed that in the realm of software defect prediction, the most frequently utilized metrics for evaluating classification algorithms are accuracy, recall, and F1-measure. These metrics are derived from the confusion matrix depicted in Table 9, where TPs represent True Positives, FPs stand for False Positives, FNs denote False Negatives, and TNs signify True Negatives. The presentation of the confusion matrix is instrumental, as the model evaluation metrics, elucidated in the ensuing subsections, are constructed based on its values. This approach ensures that optimal results align with the main diagonal of the confusion matrix.

Table 9. Structure of the Confusion Matrix

Predicted	Actual	
	Class X	Class Y
Class X	TP	FP
Class Y	FN	TN

The metrics outlined below are widely employed across various domains to assess the performance of prediction models, as detailed in [31]. **Accuracy**, represented by the ratio of both true positives (TP) and true negatives (TN) to the total number of instances examined, is a key metric. The optimal precision score is 1, while the lowest is 0. The calculation for accuracy is presented in Equation (3).

$$ACC = \frac{(TP + TN)}{TP + TN + FP + FN}$$

Precision, determined by the number of accurate positive predictions divided by the total number of positive predictions, is another essential metric. A perfect precision outcome is 1, while the least favorable is 0. Equation (4) illustrates the computation of precision, which is instrumental in gauging the effectiveness of a machine learning model in classification tasks.

$$Precision = TP/(TP + FP)$$

Recall, calculated as the number of positive predictions divided by the overall number of positives (both correctly and incorrectly classified), provides insights into the model's ability to identify

instances. The highest recall result is 1, while the lowest is 0, with Equation (5) outlining the recall calculation.

$$Recall = \frac{TP}{TP + FN}$$

F1-measure, defined as the weighted harmonic mean of precision and recall, serves as a consolidated measure combining both precision and recall for comparative analysis of different machine learning algorithms. The computation for F1-measure is depicted in Equation (6).

$$F1 - Measure = \frac{2 * Recall * Precision}{Recall + Precision}$$

In this study, the proposed algorithms were benchmarked against two widely used algorithms in the literature, namely Decision Tree and Random Forest. The evaluation considered the best results achieved by the classifiers within each dataset. The assessment results for the algorithms are presented in the subsequent sections, categorized by metric and dataset.

Accuracy Results

The accuracy outcomes are detailed in Tables 10–12, providing a comprehensive analysis of the K2, Hill Climbing, and TAN algorithms based on descriptive accuracy statistics for each dataset. Across these tables, there is minimal variability observed in the results across the ten folds. Additionally, a notable similarity is noted in the outcomes among the three algorithms.

Table 10 provides descriptive statistics for accuracy in the CM1 dataset

Algorithm	Max	Min	Mean	Median	Standard Deviation
K2	0.9183	0.898	0.9016	0.9	0.564
Hill Climbing	0.9183	0.84	0.9010	0.9	0.8353
TAN	0.92	0.76	0.8649	0.8775	4.07

Table 11 presents descriptive statistics for accuracy in the JM1 dataset

Algorithm	Max	Min	Mean	Median	Standard Deviation
K2	0.8079	0.7904	0.8043	0.8062	0.45
Hill Climbing	0.8079	0.7904	0.8043	0.8062	0.45
TAN	0.8239	0.7840	0.8063	0.8069	0.76

Table 12 displays descriptive statistics for accuracy in the KC1 dataset

Algorithm	Max	Min	Mean	Median	Standard Deviation
K2	0.8483	0.8436	0.8454	0.8436	0.22
Hill Climbing	0.8862	0.8142	0.8514	0.8530	1.53
TAN	0.8815	0.7962	0.8385	0.8388	1.88

Figures 3–5 depict the variability observed in cross-validation runs for each dataset. Figure 3 highlights the stability of accuracy in each run for the K2 and Hill Climbing algorithms. Conversely, the TAN algorithm exhibits substantial variation, indicating increased variability in results when classifying the data using this approach. In Figure 4, similar to the CM1 dataset, the JM1 dataset demonstrates minimal variability with the K2 and Hill Climbing algorithms. This suggests that these algorithms yield consistent results with the dataset, regardless of the projects used for training and testing. However, the TAN algorithm produces highly variable results. Lastly, Figure 5 reveals that, in the KC1 dataset, the TAN algorithm exhibits the most variability in accuracy. Nevertheless, there is also variability with the K2 and Hill Climbing algorithms, unlike the CM1 and JM1 datasets. Figure 5 also suggests that the choice of data for training and testing the KC1 dataset plays a significant role in achieving better or worse precision in data classification.

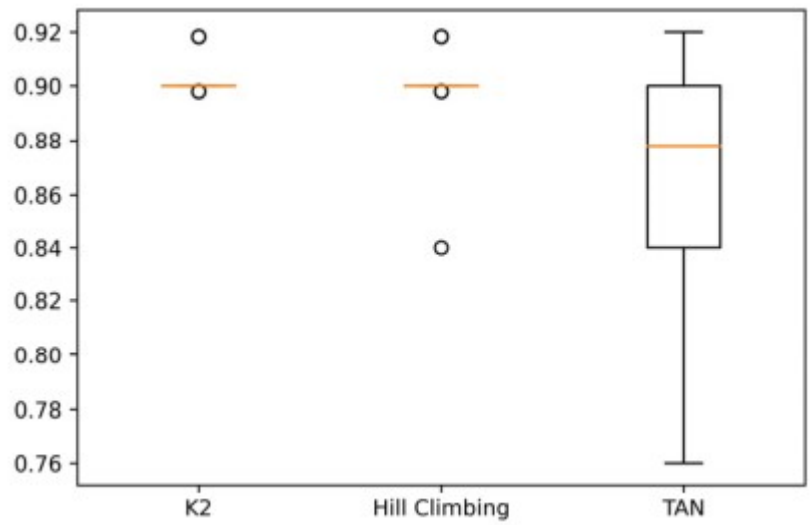

Figure 3 illustrates the fluctuation in accuracy within the CM1 dataset

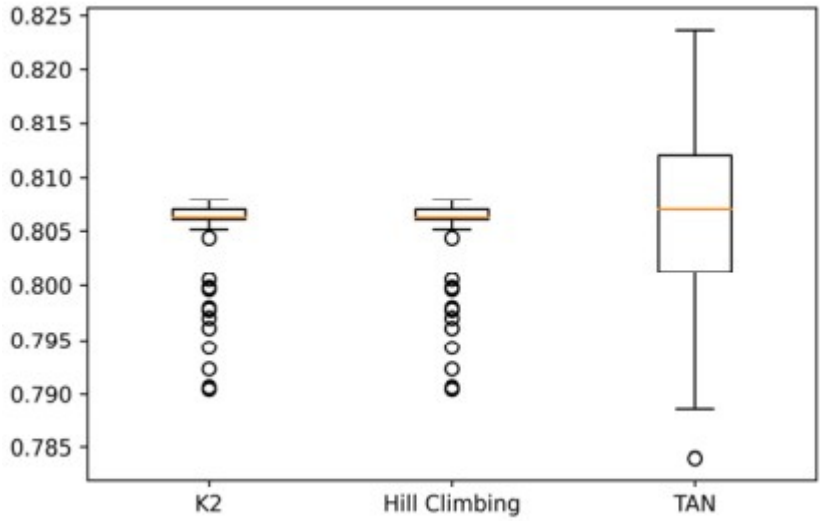

Figure 4 displays the variability in accuracy within the JM1 dataset

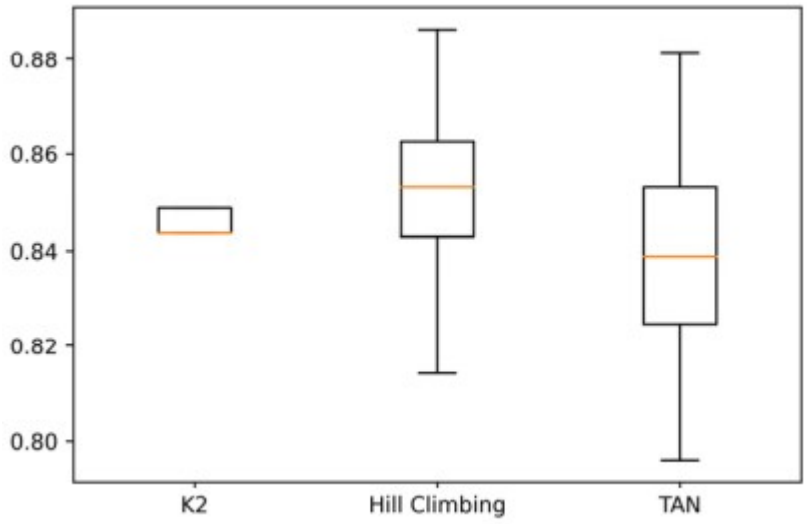

Figure 5 demonstrates the variability in accuracy within the KC1 dataset

Table 13 presents a comparison of accuracy among the proposed algorithms and other classifiers, such as Decision Tree and Random Forest. It is evident that the latter two achieved a higher accuracy percentage than the Bayesian classifiers. However, it is noteworthy that they also exhibit a higher standard deviation. This is significant as cross-validation indicates that their results display considerable variability, and their outcomes are not consistently stable. In the case of the JM1 dataset, the results by classifiers are more evenly balanced. It is important to highlight that, in this assessment, the TAN classifier achieved a higher result than the Decision Tree classifier, although Random Forest

still outperforms all. Finally, for the KC1 dataset, it is apparent that Decision Tree and Random Forest exhibit the highest accuracy.

Table 13 displays the accuracy outcomes for various datasets with alternative approaches.

Algorithm	CM1		JM1		KC1	
	Best Accuracy	Standard Deviation	Best Accuracy	Standard Deviation	Best Accuracy	Standard Deviation
K2	0.9183	0.563	0.8079	0.454	0.8483	0.225
Hill Climbing	0.9183	0.835	0.8079	0.454	0.8862	1.526
TAN	0.92	4.077	0.8236	0.767	0.8815	1.887
Decision Tree	0.94	2.865	0.8170	0.886	0.8957	1.904
Random Forest	0.94	2.084	0.8382	0.808	0.9004	1.547

Recall Outcomes

Tables 14–16 present the effectiveness of the K2, Hill Climbing, and TAN algorithms, as indicated by descriptive recall statistics for each dataset. Notably, the performance in this metric demonstrates greater stability than accuracy and yields superior results. Among these algorithms, K2 stands out as the most effective, achieving higher recall with virtually zero standard deviation.

Table 14 provides descriptive statistics for recall in the CM1 dataset

Algorithm	Max	Min	Mean	Median	Standard Deviation
K2	1	1	1	1	0
Hill Climbing	1	0.93	0.93	1	0.006
TAN	1	0.82	0.94	0.95	0.046

Table 15 presents descriptive statistics for recall in the JM1 dataset

Algorithm	Max	Min	Mean	Median	Standard Deviation
K2	1	0.92	0.98	1	0.02
Hill Climbing	1	0.92	0.98	1	0.02
TAN	0.96	0.92	0.94	0.94	0.009

Table 16 showcases descriptive statistics for recall in the KC1 dataset.

Algorithm	Max	Min	Mean	Median	Standard Deviation
K2	1	1	1	1	0
Hill Climbing	1	0.93	0.96	0.96	0.01
TAN	0.98	0.89	0.94	0.94	0.01

Figure 6 depicts boxplot diagrams illustrating recall in the CM1 dataset for each algorithm. It is observable that for K2 and Hill Climbing, no box is formed since they remain unchanged in each execution. In contrast, TAN exhibits more pronounced variability in its results, leading to the creation of a box.

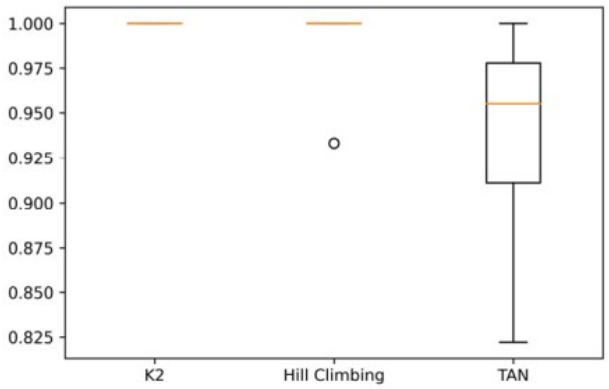

Figure 6. Fluctuations in recall within the CM1 dataset

In Figure 7, which represents the experiment conducted with the JM1 dataset, it is evident that, unlike the experiment with the CM1 dataset, the data obtained through K2 and Hill Climbing already exhibit minimal variability. Conversely, TAN continues to demonstrate more pronounced variability among its results.

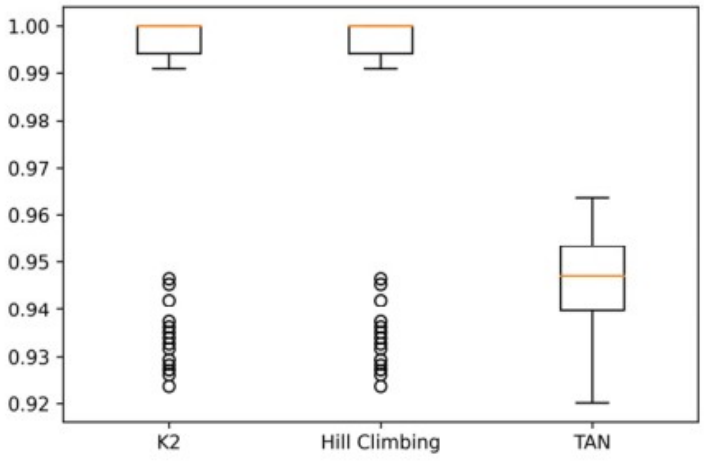

Figure 7. Fluctuations in recall within the JM1 dataset

Ultimately, Figure 8 illustrates that the K2 algorithm produces consistent results without any variations, achieving a recall of 100% (equivalent to 1). Conversely, Hill Climbing also attains 100%, but it exhibits variations in its results.

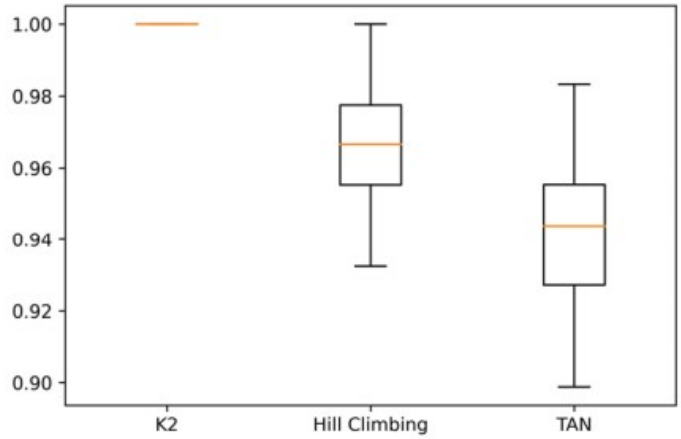

Figure 8. Fluctuations in recall within the KC1 dataset

In conclusion, the recall results from runs with the Decision Tree and Random Forest algorithms were also subject to comparison. Table 17 highlights that in the CM1 dataset, the Bayesian algorithms outperformed other approaches, displaying not only superior results but also a lower standard deviation. This indicates that their outcomes are concentrated within a narrower range of values. Notably, the K2 algorithm achieved a standard deviation value of 0, attaining a recall of 1, akin to Hill Climbing and TAN, albeit with variations in their standard deviations. In the JM1 dataset, there is some variability in the Bayesian approaches, but they achieve a higher recall than the Decision Tree and Random Forest. Lastly, they exhibit a similar pattern in the KC1 dataset, showcasing minimal variability and superior results for the Bayesian algorithms.

Table 17. Recall outcomes for various datasets with alternative approaches

Algorithm	CM1		JM1		KC1	
	Best Recall	Standard Deviation	Best Recall	Standard Deviation	Best Recall	Standard Deviation
K2	0.9183	0.563	0.8079	0.454	0.8483	0.225
Hill Climbing	0.9183	0.835	0.8079	0.454	0.8862	1.526
TAN	0.92	4.077	0.8236	0.767	0.8815	1.887
Decision Tree	0.94	2.865	0.8170	0.886	0.8957	1.904
Random Forest	0.94	2.084	0.8382	0.808	0.9004	1.547

F1-Measure Outcomes

Tables 18–20 present the effectiveness of the K2, Hill Climbing, and TAN algorithms, as indicated by descriptive F1-measure statistics for each dataset. While a value of 1 is not achieved in any case, the

low variability of this metric is evident for Bayesian algorithms. Thus, it can be concluded that in most instances, similar values will be obtained irrespective of the data used for testing or training.

Table 18 provides F1-measure descriptive statistics for the CM1 dataset.

Algorithm	Max	Min	Mean	Median	Standard Deviation
K2	0.95	0.94	0.94	0.94	0.003
Hill Climbing	0.95	0.91	0.94	0.94	0.004
TAN	0.95	0.86	0.92	0.93	0.024

Table 19 displays descriptive statistics for F1-measure in the JM1 dataset

Algorithm	Max	Min	Mean	Median	Standard Deviation
K2	0.89	0.87	0.89	0.89	0.005
Hill Climbing	0.89	0.87	0.89	0.89	0.005
TAN	0.89	0.87	0.88	0.88	0.004

Table 20 provides descriptive statistics for F1-measure in the KC1 dataset

Algorithm	Max	Min	Mean	Median	Standard Deviation
K2	0.91	0.91	0.91	0.91	0.001
Hill Climbing	0.93	0.89	0.91	0.91	0.008
TAN	0.93	0.88	0.90	0.90	0.011

Figures 9–11 illustrate the variation in F1-measure with each algorithm during the cross-validation process. Figure 9 reveals minimal variability with K2 and Hill Climbing. However, in Figure 10, while there is little variability, outliers are noticeable in K2 and Hill Climbing. Lastly, in Figure 11, although no outliers are present, K2 consistently maintains the most stable results.

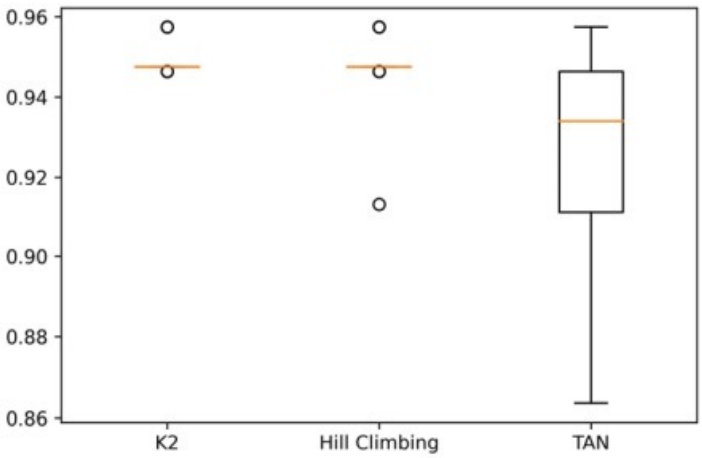

Figure 9. Fluctuations in F1-measure within the CM1 dataset.

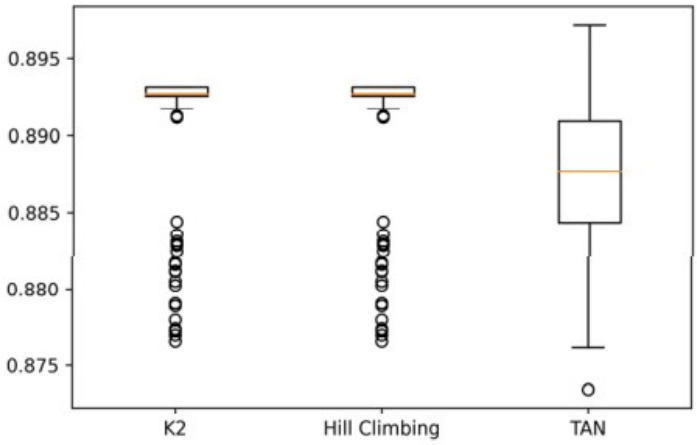

Figure 10. Fluctuations in F1-measure within the JM1 dataset

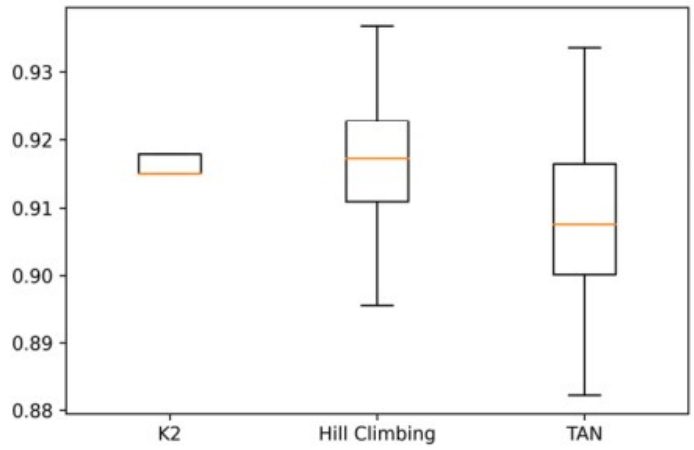

Figure 11. Fluctuations in F1-measure within the KC1 dataset

Table 21 reveals that there is not significant variation in results between the Bayesian algorithms and Decision Tree and Random Forest. However, unlike recall, none of them reaches a value of 1. Nevertheless, Bayesian algorithms, when compared to Decision Tree and Random Forest, exhibit more consistency and, consequently, less variability. This characteristic enhances the reliability of Bayesian algorithms across all datasets.

Table 21 displays the F1-measure results for various datasets using alternative approaches.

Algorithm	CM1		JM1		KC1	
	Best F1-M	Standard Deviation	Best F1-M	Standard Deviation	Best F1-M	Standard Deviation
K2	0.9574	0.003	0.8931	0.005	0.9179	0.001
Hill Climbing	0.9574	0.004	0.8931	0.005	0.9368	0.008
TAN	0.9574	0.024	0.89.72	0.004	0.9336	0.011
Decision Tree	0.9677	0.011	0.8934	0.005	0.9405	1.904
Random Forest	0.9677	0.016	0.8934	0.005	0.9405	1.547

CONCLUSIONS

This research aimed to contribute to the field of software defect prediction by exploring lesser-known classification approaches within software engineering. The utilization of Bayesian Networks was proposed as an opportunity to enhance defect prediction, as these approaches had not been extensively studied in the existing literature. The research employed cross-validation experiments to validate prediction models, and the results were compared using metrics such as accuracy, precision, recall, and F1-measure.

Upon analysis of the results, significant variability was observed among them. This variability can be attributed to the imbalanced nature of the datasets used in the evaluations—specifically, CM1, JM1, and KC1 sourced from the PROMISE repository. This imbalance introduces challenges during experimentation. Consequently, future work could explore the use of algorithms that facilitate class balancing.

The results indicate that initialization of a Bayesian Network using search algorithms like Hill Climbing and K2 yields less variability in the evaluation metric values. This suggests greater robustness and independence of the data selection process. In contrast, TAN consistently exhibited the most significant variability across all experiments. This variability is linked to the structure of the network formed from the training data. TAN utilizes a dependency tree to capture relationships between variables in the Bayesian Network, typically assuming linear dependencies. If non-linear or complex dependencies exist between variables, TAN may struggle to capture them effectively. Hence, the choice of training data significantly influences the classification results.

Based on the analysis of the obtained results, it is deduced that while the three chosen Bayesian algorithms did not significantly surpass those documented in the literature, with Random Forest exhibiting the highest performance, the metrics used yielded comparable values. Notably, Bayesian algorithms demonstrated less variability in results, rendering them robust to variations in data selection for training or testing. This characteristic contrasts with Random Forest, which relies on the selection of random samples, making it susceptible to fluctuations in accuracy.

Therefore, the recommendation is to consider employing Bayesian algorithms, as their results are comparable to Decision Tree and Random Forest, yet they offer consistent outcomes with minimal variability, irrespective of the data chosen for training or testing.

For future research, a suggestion is made to compare the results of Bayesian approaches with other ensemble classifiers beyond Random Forest. Ensemble classifiers combine multiple simpler models to achieve more accurate and robust predictions. Examples of such classifiers include Gradient Boosting, which constructs a weighted combination of weaker classification models, usually Decision Trees, and Bagging, similar to Random Forest but allowing sampling with replacement. Ensemble classification methods are advantageous for enhancing prediction accuracy by combining multiple models. Simultaneously, Bayesian Networks represent models that facilitate probabilistic reasoning by explicitly depicting relationships between variables. The selection between these approaches depends on the specific problem, available data, and analysis objectives. This proposed future work aims to assess whether investing in more complex algorithms enhances the accuracy of predicting software defects.

Finally, although this study aimed to compare the performance of various machine learning approaches (Decision Trees, ensemble algorithms, and Bayesian-based methods), these techniques could be contrasted with alternative approaches such as estimation based on analogy. In this method, previously completed projects similar to the one under estimation are selected. Another option could involve a comparison with "expert judgment," where professionals in software estimation provide estimates based on their experience.

BIBLIOGRAPHICAL REFERENCES

Meiliana, S.K., Karim, S., Warnars, H.L.H.S., Gaol, F.L., Abdurachman, E., Soewito, B. Software Metrics for Fault Prediction Using Machine Learning Approaches: A Literature Review with PROMISE Repository Dataset. In Proceedings of the 2022 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom), Phuket, Thailand, 20–22 November 2017.

Hammanouri, A., Hammad, M., Alnabhan, M., Alsarayrah, F. Software Bug Prediction using Machine Learning Approach. Int. J. Adv. Comput. Sci. Appl. 2018, 9, 78–83.

Misirli, A., Bener, A.B. A Mapping Study on Bayeasian Networks for Software Quality Prediction. In Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), Hyderabad, India, 3 June 2014.

Herzing, K., Just, S., Zeller, A. It's Not a Bug, It's a Feature: How Misclassification Impacts Bug Prediction. In Proceedings of 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 18–26 May 2013.

Hernández-Molinos, M.J., Sánchez-García, Á.J., Barrientos-Martínez, R.E. Classification Algorithms for Software Defect Prediction: A Systematic Literature Review. In Proceedings of the 2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT), San Diego, CA, USA, 25–29 October 2021.

Li, R., Zhou, L., Zhang, S., Liu, H., Huang, X., Sun, Z. Software Defect Prediction Based on Ensemble Learning. In Proceedings of 2019 2nd International Conference on Data Science and Information Technology (DSIT), Seoul, Republic of Korea, 19–21 July 2019.

Aydin, Z.B.G., Samli, R. Performance Evaluation of Some Machine Learning Algorithms in NASA Defect Prediction Data Sets. In Proceedings of the 2020 5th International Conference on Computer Science and Engineering (UBMK), Diyarbakir, Turkey, 9–11 September 2020.

Goyal, S. Heterogeneous Stacked Ensemble Classifier for Software Defect Prediction. In Proceedings of the 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC), Waknaghat, India, 6–8 November 2020.

Aljamaan, H., Alazba, A. Software Defect Prediction using Tree-Based Ensembles. In Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE), Online, 8–9 November 2020.

Ge, J., Liu, J., Liu, W. Comparative Study on Defect Prediction Algorithms of Supervised Learning Software Based on Imbalanced Classification Data Sets. In Proceedings of the 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence,

Networking and Parallel/Distributed Computing (SNPD), Busan, Republic of Korea, 27–29 June 2018.

Prahba, C.L., Shivahumar, N. Software Defect Prediction Using Machine Learning Techniques. In Proceedings of the 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 15–17 June 2020.

Ahmed, M.R., Ali, M.A., Ahmed, N., Zamal, M.F.B., Shamrat, F.M.J.M. The Impact of Software Fault Prediction in Real-World Application: An Automated Approach for Software Engineering. In Proceedings of the 2020 the 6th International Conference on Computing and Data Engineering (ICCDE), Sanya, China, 4–6 January 2020.

Nehi, M.M., Fakhrpoor, Z., Moosavi, M.R. Defects in The Next Release; Software Defect Prediction Based on Source Code Versions. In Proceedings of the Iranian Conference on Electrical Engineering (ICEE), Mashhad, Iran, 8–10 May 2018

Zhou, Y., Shan, C., Sun, S., Wei, S., Zhang, S. Software Defect Prediction Model Based On KPCA-SVM. In Proceedings of the 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), Leicester, UK, 19–23 August 2019.

El-Shorbagy, S.A., El-Gammal, W.M., Abdelmoez, W.M. Using SMOTE and Heterogeneous Stacking in Ensemble learning for Software Defect Prediction. In Proceedings of the 7th International Conference on Software and Information Engineering (ICSIE), Cairo, Egypt, 2-4 May 2018.

Bhutamapuram, U.S.; Sadam, R. Within-project defect prediction using bootstrap aggregation based diverse ensemble learning

technique. J. King Saud Univ. Comput. Inf. Sci. [Year], [Volume], [Pages].

Goyal, S. Handling class-imbalance with KNN (neighbourhood) under-sampling for software defect prediction. Artif. Intell. Rev.

[Year], [Volume], [Pages].

Malhotra, R.; Meena, S. Defect prediction model using transfer learning. Soft Comput. [Year], [Volume], [Pages].

Goyal, S. Effective software defect prediction using support vector machines (SVMs). Int. J. Syst. Assur. Eng. Manag. [Year], [Volume],

[Pages].

Cornfield, J. Bayes Theorem. Rev. De L'institut Int. De Stat. [Year], [Volume], [Pages].

Madden, M.G. On the classification performance of TAN and general Bayesian networks. In Research and Development in Intelligent

Systems XXV. SGAI 2008; Springer: London, UK, [Year]; pp. 3–16.

Friedman, N.; Geiger, D.; Goldszmidt, M. Bayesian network classifiers. Mach. Learn. [Year], [Volume], [Pages].

Gámez, J.A.; Mateo, J.L.; Puerta, J.M. Learning Bayesian networks by hill climbing: Efficient methods based on progressive

restriction of the neighborhood. Data Min. Knowl. Discov. [Year], [Volume], [Pages].

Cooper, G.F.; Herskovits, E. A Bayesian method for the induction of probabilistic networks from data. Mach. Learn. [Year], [Volume],

[Pages].

He, Y.L.; Zhao, W.J.; Xu, Y.; Zhu, Q.X. Research and Improvement of K2 Algorithm Based on Topological Sorting. In Proceedings

of the 2021 China Automation Congress (CAC), Beijing, China, [Year]; pp. 4623–4626.

Shirabad, J.S.; Menzies, T.J. The PROMISE Repository of Software Engineering Databases [Data Set]; School of Information Technology and Engineering, University of Ottawa: Ottawa, ON, Canada, [Year]; Available online: http://promise.site.uottawa.ca/

SERepository (accessed on 1 February 2022).

McCabe, T.J. A Complexity Measure. IEEE Trans. Softw. Eng. [Year], [Volume], [Pages].

Halstead, M.H. Elements of Software Science (Operating and Programming Systems Series) [Data Set]; Elsevier Science Inc.: Amsterdam,

The Netherlands, [Year].

Henry, S.; Selig, C. Predicting Source-Code Complexity at the Design Stage. IEEE Softw. [Year], [Volume], [Pages].

Fushiki, T. Estimation of Prediction Error by Using K-Fold Cross-Validation. Statics Comput. [Year], [Volume], [Pages].

Das, N.N.; Kumar, N.; Kaur, M.; Kumar, V.; Singh, D. Automated deep transfer learning-based approach for detection of

COVID-19 infection in chest X-rays. Irbm [2022], [43], [114-119].