http://doi.org/10.35784/iapgos.6393 *received: 13.07.2024 | revised: 30.10.2024 | accepted: 14.11.2024 | available online: 21.12.2024*

DEFECT SEVERITY CODE PREDICTION BASED ON ENSEMBLE LEARNING

Ghada M.T. Aldabagh, Safwan O. Hasoon
University of Mosul, Department Computer Sciences and Mathematics, Mosul, Iraq

Abstract. *In machine learning, learning algorithms that learn from other algorithms are called meta-learning. New algorithms called Ensemble algorithms have surfaced as a viable method to improve defect prediction models' accuracy and dependability. In software development defect prediction of software engineering is still a big challenge, and leads to the failure of systems, increases the cost of maintenance, and makes the development process more difficult. Consequently, defect prediction systems have become more popular as a way to foresee possible flaws early on in the development process. Defect prediction is the process that specifies the possible defects in the code written newly or the existing modified code without the use of code testing. This paper introduces ensemble learning ideas, reviews the traditional defect prediction models, and investigates ensemble learning techniques for defect classification and prediction such as bagging, boosting, stacking, and random forests, Case studies and actual experiments illustrate the important role of ensemble algorithms in classifying five severity types of defects and predicting the severity code of defects to improve the software development process by reducing the time and effort needed to determine the type of defect.*

Keywords: defect prediction, ensemble algorithm, software development, software engineering

PRZEWIDYWANIE WAGI DEFEKTU KODU NA PODSTAWIE UCZENIA ZESPOŁOWEGO

Streszczenie. *W uczeniu maszynowym algorytmy uczenia się, które uczą się na podstawie innych algorytmów, nazywane są metauczeniem. Pojawiły się nowe algorytmy zwane algorytmami Ensemble jako realna metoda poprawy dokładności i niezawodności modeli przewidywania defektów. W rozwoju oprogramowania przewidywanie defektów w inżynierii oprogramowania jest nadal dużym wyzwaniem i prowadzi do awarii systemów, zwiększa koszty utrzymania i utrudnia proces tworzenia oprogramowania. W rezultacie systemy przewidywania defektów stały się coraz bardziej popularne jako sposób przewidywania możliwych wad na wczesnym etapie procesu rozwoju. Przewidywanie defektów to proces, który określa możliwe defekty w nowo napisanym kodzie lub istniejącym zmodyfikowanym kodzie bez użycia testowania kodu. W artykule przedstawiono koncepcje uczenia się zespołowego, dokonano przeglądu tradycyjnych modeli przewidywania defektów i zbadano techniki uczenia się zespołowego do klasyfikacji i przewidywania defektów, takie jak pakowanie, wzmacnianie, układanie w stosy i lasy losowe. Studia przypadków i rzeczywiste eksperymenty ilustrują ważną rolę algorytmów zespołowych w klasyfikacji pięć typów ważności defektów i przewidywanie kodu ważności defektów w celu usprawnienia procesu tworzenia oprogramowania poprzez skrócenie czasu i wysiłku potrzebnego do określenia rodzaju defektu.*

Słowa kluczowe: przewidywanie defektów, algorytm zespołowy, tworzenie oprogramowania, inżynieria oprogramowania

Introduction

Significant obstacles to the maintainability, dependability, and quality of software systems are presented by software faults [1]. Defects must be found and fixed early in the software development lifecycle to guarantee the production of high-quality software [1].

In light of this, methods of defect prediction have become essential resources for quality assurance and software developers. These methods proactively detect any flaws in software code by utilizing machine learning models and historical data [3].

Recently, using ensemble algorithms in defect prediction become very effective and very important.

The ensemble learning algorithm combines multiple models to increase the accuracy of prediction and robustness and plays an important role in a range of fields such as natural language processing, image classification, quality assurance, etc. [4].

Researchers and practitioners have focused on investigating how ensemble approaches might be used to overcome the difficulties associated with defect identification, realizing that ensemble learning has the potential to improve defect prediction [5].

This paper delves into the convergence of defect prediction and ensemble learning, to understand how ensemble algorithms can be used in defect prediction in software engineering.

The benefits of ensemble learning algorithms like stacking, bagging, random forest, decision tree, and boosting are explained in how can make the prediction process more accurate, facilitate, and make the work of the development team more faster than using traditional methods [6].

The proposed hybrid system solve the problem of prediction and classification the degree of severity in the code to help the developer team to find the more accurate solution of the severity code, make the development process faster and make the corrected software more efficient.

This proposed study aims to classify five types of severity codes of the defect and predict the type of severity of defects for any code by using the ensemble algorithm this makes the development process more effective, accurate, and faster. The quality of the resulting software be more efficient, as well as the decision of the quality assurance team will be more accurate.

Early predicting the severity code of the software and using ensemble learning help the development team to reduce the total cost of maintenance, and enhance the software development process.

1. Related work

Defect prediction very important field in software development and ensemble learning algorithms have an important role in recent works. In recent years, extensive research has been conducted on applying ensemble learning techniques and individual machine learning models for software defect prediction. In [14] Mohammed A., Kora R. completeness performance measures are used to assess ensemble methods' prediction accuracy. The results indicate that compared to the individual fault prediction strategies under discussion, the ensemble methods that have been described produce predictions with higher accuracy. Additionally, for every dataset that was used, the results were consistent. In [3] Muhammad Azam, Muhammad Nouman, Ahsan Rehman Gill "Comparative Analysis of Machine Learning techniques to Improve Software Defect Prediction" Finding flaws utilizing the five NASA data sets JM1, CM1, KC1, KC2, and PC1 is the main problem. Among the rest, it has been demonstrated that Logistic Regression produces the greatest results (93%).

In [21] Haonan Tong, Bin Liu, and Shihai Wang "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning", present a novel SDP strategy, called SDAEsTSE, that leverages ensemble learning and SDAEs, specifically the suggested two-stage ensemble (TSE). In [11] Ran Li, Lijuan Zhou, Shudong Zhang, Hui Liu, Xiangyang Huang, Zhong Sun "Software Defect Prediction Based on Ensemble Learning", proves that the random forest is the best algorithm in defect prediction by using the comparison of experimental results, and uses the SMOTE over-sampling and Resample methods to improve the dataset's quality and improve the performance of defect classification effectively". In [2]

Abdullah Alsaeedi, Mohammad Zubair Khan, "Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study", examines and contrasts the ensemble classifiers and supervised machine learning techniques using ten NASA datasets. Based on the experimental data, found that RF outperformed the other classifiers in most of the scenarios. In [16] Sushant Kumar Pandey, Ravi Bhushan Mishra, Anil Kumar Tripathi, "BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques", demonstrate the effective outcome of using machine learning and feature selection techniques to distinguish between problematic software modules and those that are not. For SBP, suggest a basic classification-based framework called Bug Prediction that makes use of Deep representation and Ensemble learning (BPDET) approaches. In [18] Santosh S. Rathore, Sandeep Kumar (2020),” An empirical study of ensemble techniques for software fault prediction” investigated ensemble approaches for SFP are. Seven ensemble strategies are evaluated empirically: Dagging, decorating, Grading, MultiBoostAB, RealAdaBoost, Rotation Forest, and Ensemble Selection.

In [10] Hemant Kumar, Vipin Saxena "Software Defect Prediction Using Hybrid Machine Learning Techniques: A Comparative Study" the suggested method combines ensemble models like Support Vector Machine (SVM), Random Forest (RF), and XGBoost with an advanced deep neural network architecture. The PROMISE Software Engineering Repository's datasets, together with those from several software projects such as CM1, JM1, KC1, and PC1, are used in the study's evaluation of performance. By giving a comparative viewpoint on early defect identification and mitigation tactics, the study that is being presented provides insightful information about the efficacy of hybrid methodologies for cross-project defect prediction.

2. Software defects

2.1. Defect types

Bugs or software defects can be defined as abnormal or flaws in software design or code and cause erroneous behavior, malfunctions in the system, or other problems. Defect prediction, quality control, and software development all depend on an understanding of various defects kinds. The specific types of defects as shown in Table 1 and describe below [7].

Syntax errors

One of the simplest and most common types of errors in software development is syntax errors. They arise from code that deviates from the conventions and grammar of the programming language being used. Missing semicolons, mismatched parenthesis, and typographical errors are a few examples [7].

Logic errors

Semantic mistakes, also referred to as logic errors, happen when a piece of code runs correctly in terms of syntax but fails to generate the desired effects because of faulty reasoning. The fact that the code functions without producing error signals makes it difficult to find these flaws. Logic problems are frequently found using testing and debugging techniques [7].

Runtime errors

Runtime errors can result in program crashes or undesirable outcomes when they occur during program execution. Division by zero, null pointer exceptions, and array index out-of-bounds errors are common instances. Effective error-handling techniques are essential for reducing the effects of runtime errors [7].

Boundary conditions

When inputs are close to specific borders or limits, the code may react differently. This is known as a boundary condition flaw. Unexpected behavior can arise, for instance, while processing a variable's maximum or lowest permissible value due to a bug [7].

Data type errors

When one type of data is incorrectly handled as another, an error occurs called a data type error. These flaws may result in inaccurate computations or tampered data. Data type errors might arise, for example, if you attempt to apply mathematical operations on strings as opposed to numbers [7].

Resource leaks

When a program keeps using memory, files, or network connections after they are no longer required, it is known as a resource leak. System instability and performance deterioration might result from frequent resource leaks [7].

Concurrency and synchronization issues

Deadlocks, Race situations, and data corruption can result from thread management, synchronization, and flaws in concurrent or multi-threaded software. Careful design and testing are frequently necessary to find and address these flaws [7].

Input validation and sanitization

Software that fails to correctly validate user inputs is said to have input validation defects.

Security flaws like injection attacks or data breaches (cross-site scripting or SQL injection) may arise from this [7].

Memory management issues

Buffer overflows, which occur when data exceeds the allocated buffer size, and memory leaks, which occur when allocated memory is not deallocated, are examples of memory-related faults. These flaws may result in system failures, security openings, or unpredictability [7].

Compatibility and platform-specific issues

Due to variations in operating systems, hardware, or dependencies, software may display bugs on particular platforms or combinations. In order to guarantee that software operates successfully in a variety of contexts, compatibility flaws must be fixed [7].

Documentation defects

Errors and inconsistencies in user manuals, technical documentation, and comments found in the documentation of software are referred to as documentation defects. Understanding and maintaining software depends on accurate and current documentation [7].

Table 1. Defect classes with their severity code and descriptions

Defect type	Severity code	Description
Syntax Errors	Trivial	Has little effect on the functionality of the code
Documentation Defects	Normal	Documentation problems or other non-essential elements that don't immediately affect functionality
Boundary Conditions	Major	Serious problems that impair operation but do not result in system failures or security flaws
Data Type Errors	Major	
Performance and Efficiency Issues	Major	
Usability and User Experience Issues	Major	
Logic Errors	Critical	Critical issues can cause data corruption, system crashes, security flaws, or a severe decline in performance.
Runtime Errors	Critical	
Concurrency and Synchronization Issues	Critical	
Input Validation and Sanitization	Critical	
Memory Management Issues	Critical	
Security Vulnerabilities	Blocking	Refers to flaws that entirely stop the software from operating as intended or that obstruct important functions or functionalities. A problem that is categorized as "blocking" is the most severe kind of issue and needs to be addressed right away by the team of development.
Compatibility and Platform-Specific Issues	Blocking	
Resource Leaks	Blocking	

2.2. Severity codes

Severity codes in software defects refer to a system of classification used to categorize the effect or severity of a defect on the software system. during the development and testing process Severity codes will help to prioritize which defects should be addressed first. The severity code assigned to a defect typically indicates how severely the defect affects the functionality, performance, or usability of the software. Severity codes offer a uniform structure for managing and prioritizing defects at different stages of the software development lifecycle. This helps to make sure that blocking or critical defects are effectively addressed and managed, while trivial defects are effectively addressed and managed within the scheduled time of development [8].

2.3. Methods of defect prediction

The basics of early defect prediction approaches were single-machine learning models like decision trees and logistic regression, as well as conventional statistical techniques. These techniques frequently built prediction models using modification history, code metrics, and other software properties. These methods have difficulties managing the complexity and heterogeneity of software data, although they produce insightful results [9].

3. Ensemble algorithms

Bagging, Gradient Boosting, AdaBoost, and Random Forest, stacking were proven as effective methods in the process of predicting bugs (defects). A variety of the base learners or classifiers which is used in ensemble algorithms will increase the robustness, and accuracy of prediction defects.

Researchers have proved that an ensemble algorithm can manage unbalanced or noising datasets. As a result, these methods will enhance the process of defect detection or prediction early and will improve the software development process.

Advantage of ensemble methods

Ensemble algorithms have many advantages that make the process of defect prediction, more accurate and faster some of these advantages are (more accuracy, importance features, robustness, and interpretability of the model).

Ensemble Algorithms Challenges

Despite the positive results, there are still a lot of difficulties and unanswered issues with employing ensemble algorithms for defect prediction.

Interpretability

While numerous ensemble approaches make models interpretable, some, like ad-boost, and XGBoost algorithms, might be more complex and challenging to grasp. Further study is needed to increase the ensemble models' comprehensibility and transparency.

Data imbalanced

It's still difficult to handle unbalanced datasets in defect prediction. Research is still being done to make sure the ensemble model accurately predicts uncommon flaws while minimizing the rate of high false positives. Scalability: when ensemble methods were applied to big projects this made scalability very difficult [12].

4. Methodology

Defect prediction is important and defect severity prediction is most important to make the process of software development faster and more accurate all previous work was operated to detect whether there are defects or not or to classify the software as a defect or not, the main problem which solved by this wok was how to predict five types of severity code. The proposed novel work predicts five types of severity code of the defect by using ensemble learning algorithms bagging, random forest stacking, and XGBoost, can be applied these algorithms on six datasets (eclipse, free desktop, gcc, gnome, Mozilla, and winehq) results of proposed system was very accurate. The framework of the proposed system is shown in Figure 1 and a block diagram is shown in Figure 2.

The proposed system has phases explained below:

Phase one: initialize dataset the proposed study applied on six dataset datasets (eclipse, free desktop, gcc, gnome, Mozilla, and winehq).

Phase two: preprocessing in this phase all non-values are removed by using one of the methods which use previous data, the mean of the data, the most frequent data, or the median value.

Phase three: severity code this phase studies the severity code and uses ensemble learning algorithms (random forest, bagging, XGBoost, stacking for classifying the severity).

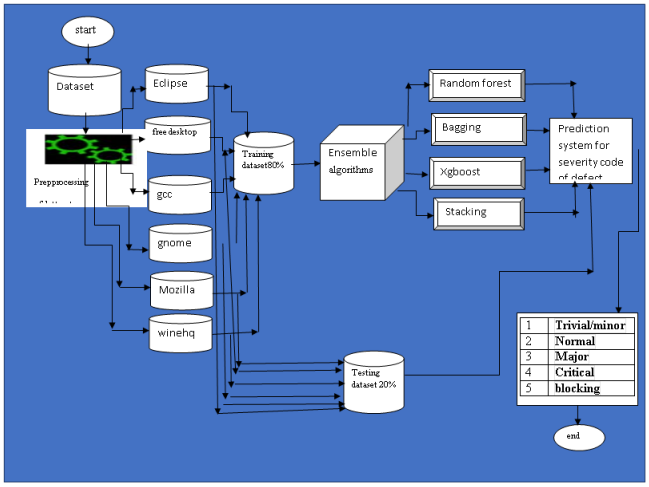


Fig. 1. Demonstrates the proposed framework for prediction of the severity code by using ensemble algorithms

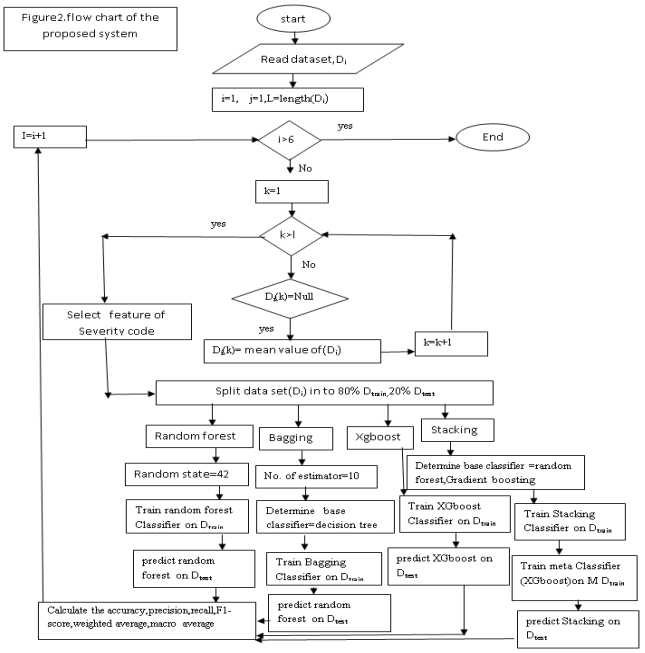


Fig. 2. Flow chart of the proposed system

4.1. Ensemble techniques

Four ensemble techniques (random forest, bagging, XGBoost, and stacking)are used to predict five severity codes of defects, a brief description of these algorithms is given as follows:

- Random forest: an effective and popular ensemble learning approach for classification, regression, and other applications is called Random Forest. During training, it creates a large number of decision trees, and then outputs the mean prediction for regression or the class for classification based on the individual trees [10].

- Bagging: bootstrap Aggregating, or Bagging, is a strategy for ensemble learning that aims to increase the accuracy and stability of machine learning models. It helps keep models from overfitting and lowers variance, especially in high-variance models like decision trees [10].
- XGBoost: with exceptional accuracy and speed, XGBoost is a machine learning algorithm that is both strong and adaptable. Its performance benefits and capacity to manage big datasets effectively make it especially well-suited for structured or tabular data, and it is frequently utilized in a variety of predictive modeling applications [10].
- Stacking: by merging several base models via a meta-model, stacking – a potent ensemble learning technique – improves prediction performance. With cross-validation, it minimizes overfitting and makes use of the advantages of many models, making it a versatile and efficient method for both regression and classification applications. When compared to other ensemble approaches, it is more sophisticated and computationally demanding [10].

4.2. Performance evaluation measures

Six performance measures precision, recall, F1-score, average weight, accuracy, macro average are used to assess the performance of all four ensemble algorithms these measures are explained as follows:

- precision: it is used to find the portion of the correctly predicted faulty modules out of all modules. It is calculated by Equation (1).

$$\text{precision} = \frac{TP}{TP+FP} \quad (1)$$

where: TP = true positive, FP = false positive.

- Recall: it is used to calculate the correct faulty models are predicted. It is calculated by Equation (2).

$$\text{recall} = \frac{TP}{TP+FN} \quad (2)$$

where: TP = true positive, FP = false positive.

- F1-score: When evaluating a classification model's efficacy, especially in situations where datasets are unbalanced, the F1-score offers a single metric that strikes a compromise between precision and recall. The F1-score provides a more complete view of a model's performance than precision or recall alone because it takes into account both erroneous positives and false negatives. It is calculated by Equation (3)

$$\text{F1-score} = \frac{precision*recall}{precision+recall} \times 2 \quad (3)$$

- macro average: A multi-class classification model's overall performance can be assessed using the macro average measure, which averages the performance measures determined separately for each class. Regardless of a class's size in the dataset, this method assigns it the same weight and treats all classes identically.it is calculated by taking the arithmetic mean of the precision, recall, or F1-score for all classes as follows.

$$\text{Macro precision} = \frac{1}{N}\sum_{i=1}^{N} precision_i \quad (4)$$

$$\text{Macro recall} = \frac{1}{N}\sum_{i=1}^{N} recall_i \quad (5)$$

$$\text{Macro F1-score} = \frac{1}{N}\sum_{i=1}^{N} F1\_score_i \quad (6)$$

- weighted average: it is a measure that is used to calculate the overall performance of a multiclassification model this measure uses the averaging of the metrics which is used to assess the performance of each class, each class contributing proportionally according to its size in the dataset. This metric is used for the imbalanced in the distribution of each class by assigning higher weight to classes which have more instances. It is calculated by following the equation.

$$\text{Weighted precision} = \frac{1}{N}\sum_{i=1}^{N} precision_i \times support_i \quad (7)$$

$$\text{Weighted recall} = \frac{1}{N}\sum_{i=1}^{N} recall_i \times support_i \quad (8)$$

$$\text{Weighted F1-score} = \frac{1}{N}\sum_{i=1}^{N} F1\_score_i \times support_i \quad (9)$$

5. Results and analysis

This section explain the results of various performance measures used by ensemble algorithms to predict the five severity code. Tables 2–5 and Figures 3–6 show the results of applying the algorithms discussed on the six data sets. Also analysis of the results is made to make clear observations about the ensemble techniques performance are explained in table 6 and Fig. 7. At last comparison the proposed system with other researcher are explained in Figure 8 and Table 7.

Table 2. Experimental results of applying random forest algorithm on six dataset

ensemble algorithm	dataset	severity code	precision	recall	F1-score	accuracy
Random forest	Eclipse	1	1.00000	0.92	0.96	0.9991364
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Free desktop	1	1.00000	0.94	0.97	0.9995676
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	GCC-bug-report	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Gnome	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Mozila	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Winehq	1	1.00000	1.00000	1.00000	0.9968944099378
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	0.625000	0.833333	0.714286	
		6	0.900000	0.750000	0.818182	

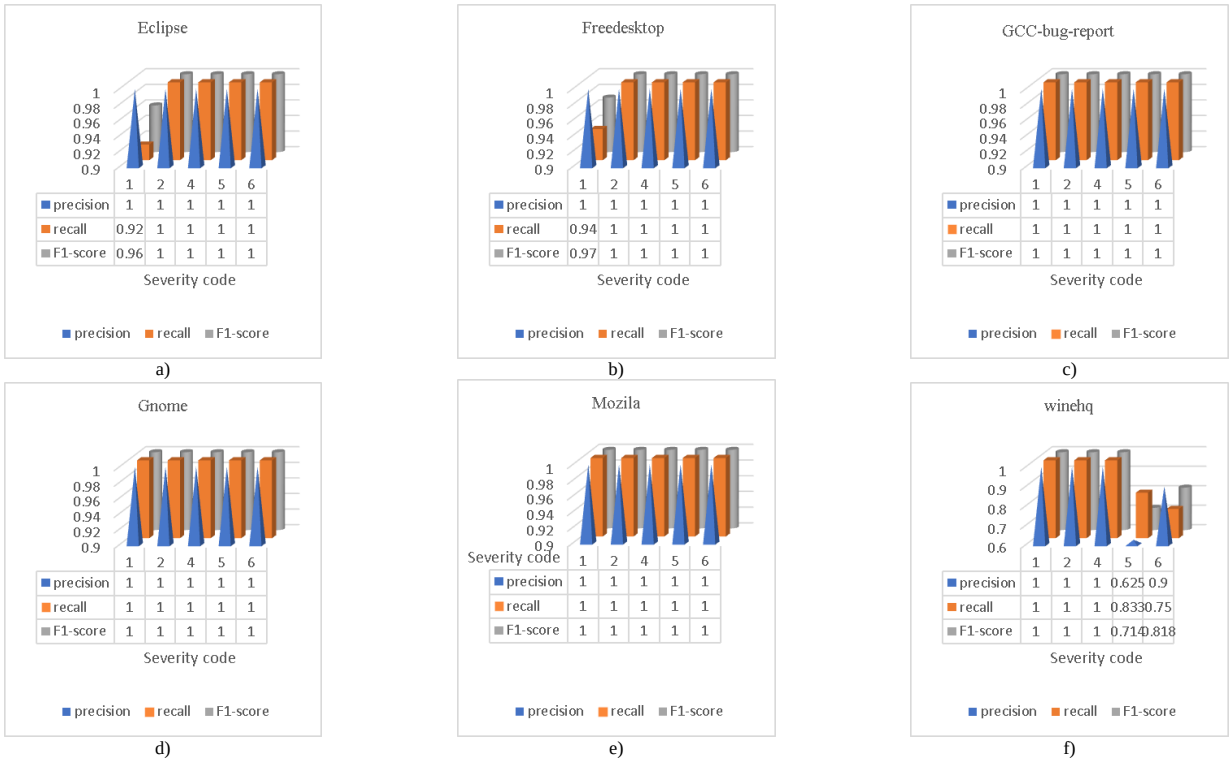


Fig. 3. Random forest experimental results on six datasets: a) Eclipse, b) Free desktop, c) GCC-bug-report, d) Gnome, e) Mozila, f) winehq datasets

Table 3. Experimental results of applying the bagging algorithm on six dataset

ensemble algorithm	dataset	severity code	precision	recall	F1-score	accuracy
Bagging	Eclipse	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Free desktop	1	1.000000	1.00000	1.00000	1.00000
		2	1.00000	1.000000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	GCC-bug-report	1	1.000000	0.750000	0.857143	0.999961331
		2	0.999961	1.000000	0.999981	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Gnome	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Mozila	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Winehq	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	

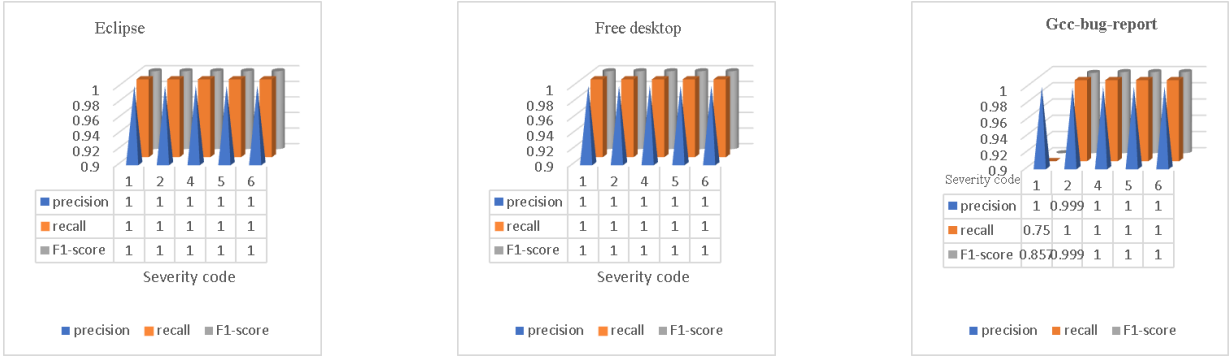
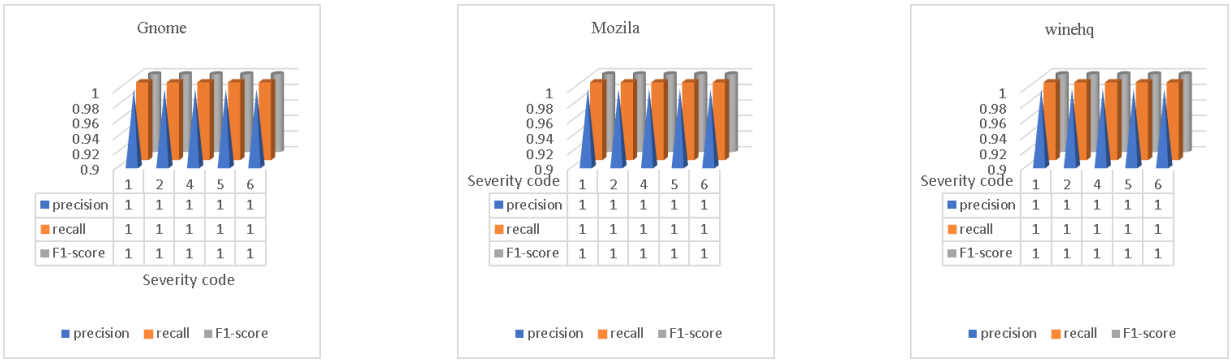


Fig. 4 (part 1). Bagging experimental results on six datasets: a) Eclipse, b) Free desktop, c) GCC-bug-report, d) Gnome, e) Mozila, f) winehq datasets



d) e) f)

Fig. 4 (part 2). Bagging experimental results on six datasets: a) Eclipse, b) Free desktop, c) GCC-bug-report, d) Gnome, e) Mozila, f) winehq datasets

Table 4. Experimental results of applying the XGBoost algorithm on six dataset

ensemble algorithm	dataset	severity code	precision	recall	F1-score	accuracy
XGBoost	Eclipse	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Free desktop	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	GCC-bug-report	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Gnome	1	1.000000	0.968750	0.984127	0.99983119513842
		2	0.999826	1.000000	0.999826	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Mozila	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Winehq	1	0.979167	1.000000	0.989474	0.9992236024844
		2	1.000000	0.999154	0.999577	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	

a) b) c)

d) e) f)

Fig. 5. XGBoost experimental results on six datasets: a) Eclipse, b) Free desktop, c) GCC-bug-report, d) Gnome, e) Mozila, f) winehq datasets

Table 5. Experimental results of applying the XGBoost algorithm on six dataset

ensemble algorithm	dataset	severity code	precision	recall	F1-score	accuracy
Stacking	Eclipse	1	1.00000	1.00000	1.00000	0.9994242947610823
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	0.97	0.98	
	Free desktop	1	1.00000	1.00000	1.00000	0.9994242947610823
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	0.96	0.97	
	GCC-bug-report	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Gnome	1	1.000000	1.000000	1.000000	1.000000
		2	1.000000	1.000000	1.000000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	Mozila	1	1.00000	1.00000	1.00000	1.00000
		2	1.00000	1.00000	1.00000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	
	winehq	1	1.000000	1.000000	1.000000	1.000000
		2	1.000000	1.000000	1.000000	
		4	1.00000	1.00000	1.00000	
		5	1.00000	1.00000	1.00000	
		6	1.00000	1.00000	1.00000	

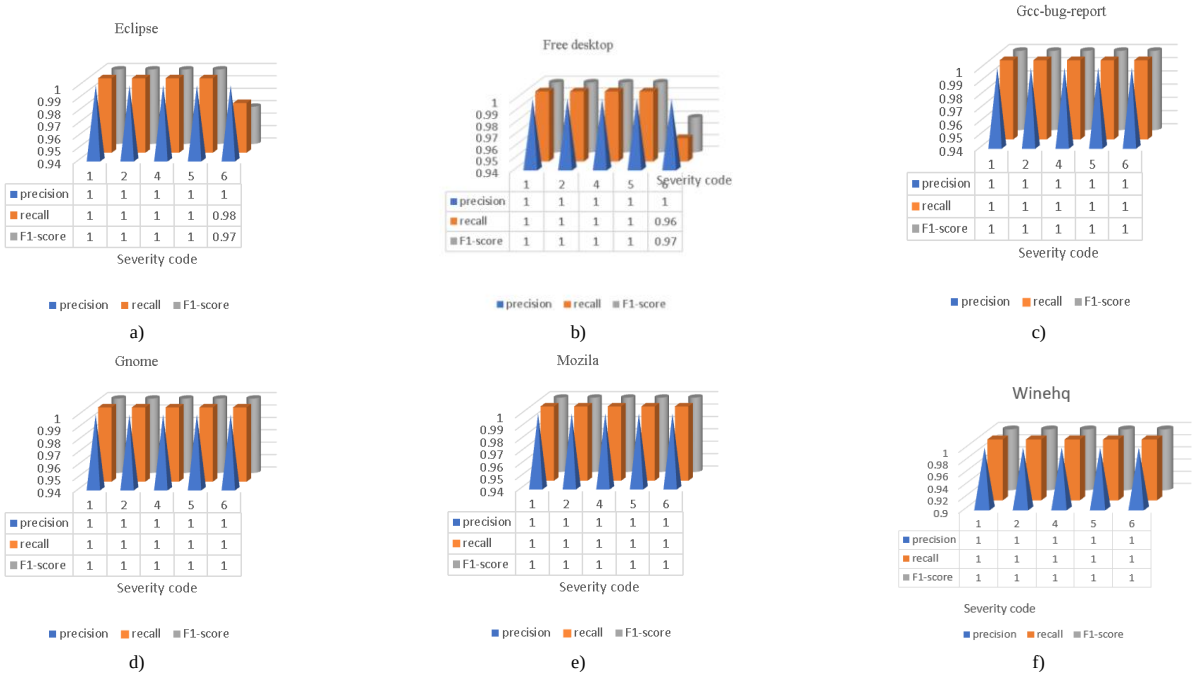


Fig. 6. Stacking experimental results on six datasets: a) Eclipse, b) Free desktop, c) GCC-bug-report, d) Gnome, e) Mozila, f) winehq datasets

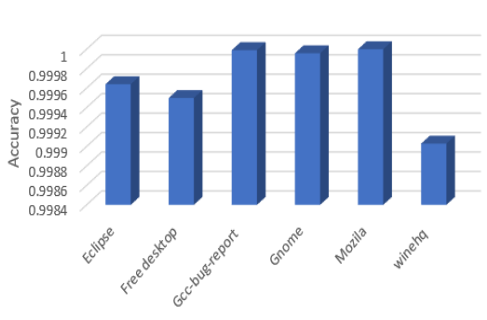


Fig. 7. Comparison the accuracy of applying proposed model on six datasets

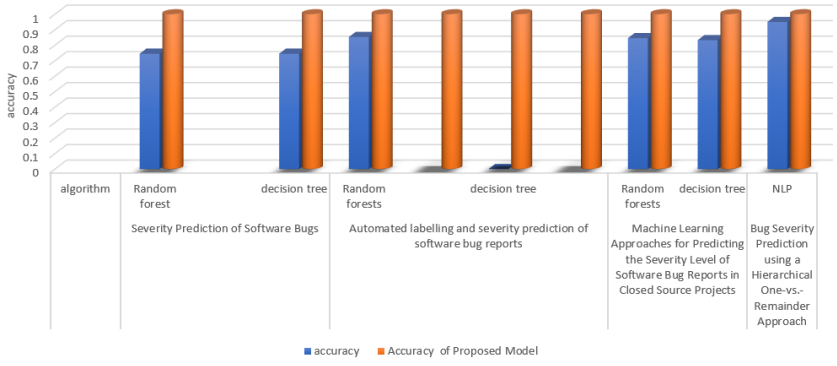


Fig. 8. The comparison results of proposed system with other researcher

Table 6. Results of the accuracy metrics of applying the four algorithms(random forest, bagging, XGBoost, stacking) on the six datasets

data set	random forest accuracy	bagging accuracy	XGBoost accuracy	stacking accuracy	accuracy average of proposed model
Eclipse	0.9991364	1.00000	1.00000	0.9994242947610823	0.999640173
Free desktop	0.9995676	1.00000	1.00000	0.9984242947610823	0.9994979736902706
GCC-bug-report	1.00000	0.999961331	1.00000	1.00000	0.99999033275
Gnome	1.00000	1.00000	0.99983119513842	1.00000	0.99995779878460
Mozila	1.00000	1.00000	1.00000	1.00000	1.00000
winehq	0.9968944099378	1.00000	0.9992236024844	1.00000	0.99902950310555

Table 7. Comparing accuracy of results of proposed model with other researcher

paper title	researcher	classification algorithm	accuracy	accuracy of proposed model
Severity Prediction of Software Bugs	Ahmed Fawzi Otoom, Doaa Al-Shdaifat, Maen Hammad, Emad E. Abdallah	Random forest	0.745	0.999640173
		Decision tree	0.745	0.999640173
Automated labelling and severity prediction of software bug reports	Ahmed Fawzi Otoom, Doaa Al-Shdaifat, Maen Hammad, Emad E. Abdallah, Ashraf Aljammal	Random forest	0.853	0.999640173
				0.999640173
		Decision tree	0.804	0.999640173
				0.999640173
Machine Learning Approaches for Predicting the Severity Level of Software Bug Reports in Closed Source Projects	Aladdin Baarah, Ahmad Aloqaily, Zaher Salah, Mannam Zamzeer, Mohammad Sallam	Random forest	0.8455	0.99999033275
		Decision tree	0.8326	0.999640173
Bug Severity Prediction using a Hierarchical One-vs.-Remainder Approach	Nonso Nnamoko Luis Adri´an Cabrera-Diego, Daniel Campbell, Yannis Korkontzelos	NLP	0.95	0.99999

6. Conclusion

This paper proposed model consisting of four ensemble algorithms to predict the five of severity code of the software.

The main objective of this research was to make the process of prediction accurate the severity code of the defect code more and make the process of developing software in term less time consumeing and efforts.

The proposed model has been applied to six datasets (Eclipse, Free desktop, GCC-bug-report, Gnome, Mozila, and winehq). These datasets are varies based on the number of projects, number of severity code, and severity code ratio. Each Experimental has used the proposed model which consists of four ensemble algorithms(Random forest, Bagging, XGBoost, and stacking)on each one of the datasets to show the performance of the proposed model based on the six evaluation metrics (Accuracy, precision,Recall, F1-score, Macro average, Weighted Average).

The experimental results have shown that the proposed system has produced the highest prediction accuracy of severity code. The proposed model goes one better than stacking and R.F. in prediction trivial and normal severity code, also goes better than bagging in prediction minor severity, and is better than XGBoost in prediction major and blocking severity code. Also, it showed the performance of each method depends on the dataset and the used classifier, as well as the proposed system outperformed previous systems proposed by other researchers, as shown in Table 7 and Figure 8. The results show that the proposed system outperformed all previous researchers.

References

[1] Alnaish Z. A. H., Hasoon S. O.: Hybrid binary whale optimization algorithm based on taper shaped transfer function for software defect prediction. Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska 13(4), 2023, 85–92 [https://doi.org/10.35784/iapgos.4569].

[2] Alsaeedi A., Khan M. Z.: Software defect prediction using supervised machine learning and ensemble techniques: a comparative study. Journal of Software Engineering and Applications 12(5), 2019, 85–100 [http://www.scirp.org/journal/jsea].

[3] Azam M., Nouman M., Gill A. R.: Comparative Analysis of Machine Learning techniques to Improve Software Defect Prediction. KIET Journal of Computing & Information Sciences – KJCIS 5, 2022, 41–66 [https://doi.org/10.51153/kjcis.v5i2].

[4] Bakhur N.: What Causes Software Bugs,Types of Defects in Software Testing, May 21, 2024, [https://neklo.com/blog/what-causes-software-bugs].

[5] Brazdil P. et al.: Metalearning: applications to automated machine learning and data mining. Springer Nature, 2022.

[6] Brownlee J.: What are the Benefits of Ensemble Methods for Machine Learning. 2021 [https://machinelearningmastery.com/why-use-ensemble-learning].

[7] Chmielowski L., Kucharzak M., Burduk R.: Application of Explainable Artificial Intelligence in Software Bug Classification. Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska 13(1), 2023, 14–17 [https://doi.org/10.35784/iapgos.3396].

[8] Dada E. G. et al.: Advances in Machine Learning & Artificial Intelligence, 2021 [http://www.opastonline.com].

[9] Haldar S., Capretz L. F.: May. Explainable Software Defect Prediction from Cross Company Project Metrics using Machine Learning. 7th International Conference on Intelligent Computing and Control Systems (ICICCS). India, Madurai, 2023, 150–157 [https://doi.org/10.1109/ICICCS56967.2023.10142534].

[10] Kumar H., Saxena V.: Software Defect Prediction Using Hybrid Machine Learning Techniques: A Comparative Study. Journal of Software Engineering and Applications 17(4), 2024, 155–171.

[11] Li R. et al.: Software defect prediction based on ensemble learning. International conference on data science and information technology. USA, New York, NY, 2019, 1–6 [https://doi.org/10.1145/3352411.3352412].

[12] Matloob F. et al.: Software defect prediction using ensemble learning: A systematic literature review. IEEE Access 9, 2021, 98754–98771.

[13] Mienye I. D. Sun Y.: A survey of ensemble learning: Concepts, algorithms, applications, and prospects. IEEE Access 10, 2022, 99129–99149 [https://creativecommons.org/licenses/by/4.0/99].

[14] Mohammed A., Kora R.: A comprehensive review on ensemble deep learning: Opportunities and challenges. Journal of King Saud University-Computer and Information Sciences 35(2), 2023, 757–774.

[15] Olaleye, T. O. et al.: Predictive analytics and software defect severity: A systematic review and future directions. Scientific Programming 1, 2023, 6221388 [https://doi.org/10.1155/2023/6221388].

[16] Pandey S. K., Mishra R. B., Tripathi A. K.: BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. Expert Systems with Applications 144, 2020, 113085 [https://doi.org/10.1016/j.eswa.2019.113085].

[17] Prabha C. L. Shivakumar N.: Software defect prediction using machine learning techniques. 4th International Conference on Trends in Electronics and Informatics (ICOEI) (48184). India, Tirunelveli, 2020, 728–733.

[18] Rathore S. S., Kumar S.: An empirical study of ensemble techniques for software fault prediction. Applied Intelligence 51, 2021, 3615–3644.

[19] Rathore S. S., Kumar S.: Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems. Knowledge-Based Systems 119, 2017, 232–256 [https://doi.org/10.1016/j.knosys.2016.12.017].

[20] Tang Y. et al.: Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm. International Journal of Machine Learning and Cybernetics 14(6), 2023, 1967–1987.

[21] Tong H., Liu B., Wang S.: Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. Information and Software Technology 96, 2018, 94–111 [https://doi.org/10.1016/j.infsof.2017.11.008].

M.Sc. Ghada M.T. Aldabagh
e-mail: ghadaaldabagh@uomosul.edu.iq

She has a master's in Computer Science from the University of Mosul, Currently,she works as a lecturer in the College of Computer Science and Mathematics, University of Mosul, Mand a Ph.D. candidate in the Computer Science Department, College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq.
Research interest: machine learning, metalearning artificial intelligence, artificial neural network, software engineering.

https://orcid.org/0000-0002-4673-2288

Prof. Safwan O. Hasoon
e-mail: Dr.safwan1971@uomosul.edu.iq

He is a doctor and full professor of artificial intelligence. He works at the College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq.
Research interest: machine learning, artificial intelligence, artificial neural network

https://orcid.org/0000-0002-3653-3568