Bug Hierarchy Optimization Using Meta-Heuristic Algorithms

Hardik Nahta, Aayush Agarwal, Vinay Malik and Jayesh Bakle
Department of Computer Science and Engineering
Indian Institute of Information Technology, Kota
Kota 325003, Rajasthan, India
{hardiknahta111, aayushagarwal7775, vinaymalik1729, jayeshbakleabd}@gmail.com

Dr. Ajay Nehra
Department of Computer Science and Engineering
Indian Institute of Information Technology, Kota
Kota 325003, Rajasthan, India
ajay.cse@iiitkota.ac.in

***Abstract*—It takes a lot of effort and time to manually assign bugs the correct prioritization. Currently, only one feature is used to prioritize issues, which results in data loss because bugs depend on many other features. In this paper, we proposed an efficient way of selecting bug features for prioritizing bugs more accurately using Particle Swarm Optimization and Genetic Algorithm. First, we preprocess the textual data from bug reports using natural language processing (NLP) techniques, and then we use term frequency inverse document frequency to convert the textual features into numerical features. Numerous additional features are produced during this translation, increasing the algorithms' complexity and execution time. We choose the most relevant and practical feature set using PSO and GA in order to further reduce this. Our suggested model combines classification algorithms with feature reduction techniques. For experiments, Eclipse, Mozilla, Thunderbird, and Firefox datasets are used. The model performs better, according to the results, both with all features and with fewer features, in terms of recall, precision, and accuracy. With fewer features, we were able to obtain improvement (using PSO and GA). The findings show that, in terms of average accuracy, the GA-KNN approach outperforms the PSO-KNN method.**

***Index Terms*—PSO (Particle Swarm Optimization), text categorization, bug prioritization, NLP (Natural Language Processing).**

I. INTRODUCTION

Bug prioritization is crucial in software development to fix defects efficiently and ensure high-quality software products. Manual bug prioritization processes are often difficult and can lead to biases, especially in large-scale software projects. To solve these bug issues, developers collect feedback and reviews from end users to resolve the issues that they faced while using software, but they don't easily fix bugs for years and take long time due to various factors such as time it takes and availability of suitable developers. All these processes consumes available time and a lot of human efforts and sometimes generate incorrect results.

ML (Machine Learning) and DL (Deep Learning) classifiers each have their own drawbacks when prioritizing bugs, and performing text classification process. (SVM) Support vector machine is one of the best ML algorithm that requires major feature modeling efforts many a times. Conversely, CNN (Convolutional Neural Network) automatically extracts patterns in features of bugs and many position-invariant features independently but also removes the need for feature modeling in order to rank bugs that SVM was unable to do. These capabilities such as controlling position invariant features and pattern matching are very effective in emotion analysis of reviews provided by end users. For optimal performance of CNN we require tuning hyperparameter settings so that overfitting can be avoided, hence this all requires a deep and knowlegable understanding of both CNN and the problem to be addressed with CNN.

In this context, research paper explores the development and implementation of a bug prioritization system using meta-heuristic feature selection algorithms. The main goal is to automate the bug prioritization process, ensuring that critical bugs are identified and addressed properly on time based on their severity and potential impact. By using algorithms like GA and PSO, our proposed system aims to identify optimal features within bug datasets, provide accurate bug prioritization.

This includes a feature weighting method that focuses on PSO and GA to find relevant features/property for finding appropriate severity. Our proposed model also uses classification algorithms such as KNN to predict different levels of bug severity in software. Datasets are used to finally evaluate the performance of the proposed model, which involved predicting bug severity levels in software using PSO-KNN and GA-KNN methods. Overall, a bug prioritization system using meta-heuristic feature selection algorithms is an effective approach to improving software development practices, and has the potential to become an essential tool for software development teams.

A bug priority system holds numerous and essential roles in software development techniques. By through identifying and setting the priorities of the defects, developers can devote their resource more appropriately, fixing the key bugs firstly. This doubles up as software quality improves and to tackle productivity & maintenance cost. The research objectives requires the following steps:

- Collection of bug dataset of 4 different open source softwares that involve the level of severity, root cause finding, the budget and other textual features.
- Design and implementation of the bug prioritization system using meta-heuristic feature selection algorithms.
- System efficiency evaluation involves the application of different metrics, comprising of F1 score, accuracy, recall, and precision
- Comparison of he suggested strategy to manual and other automated methods for predicting problem priority.
- Analysis of results to identify strengths, weaknesses, and areas for improvement.

Bug priority is significantly impacted by the suggested paradigm. It is useful for open source software such as the ones used in the research done by us, i.e., Eclipse, Thunderbird, Mozilla, and Firefox, to prioritize correcting significant defects and has improved the prediction accuracy of bugs. Software companies can increase the quality of their products by implementing the suggested model.

This research article's remaining sections are arranged as follows. A review of the literature and an outline of earlier research developments are given in Section II. This chapter covers the various methods that researchers employ. Evaluation criteria and a review of related published material are also covered. In Section III, the research technique is presented. The first section outlines the procedures for analyzing the current bug priority model, identifying flaws in the current approach, and suggesting enhancements. Phase two will include suggested enhancements to the current system. The experimental results of the suggested system are shown and discussed in Section IV, along with a comparison of the findings with previous methodologies. This paper finishes with contributions and aims that have been achieved in Section V.

II. RELATED WORK

A. Background

In software engineering, the constant version updates of software components requires deep testing before deployment. Among the processes to smooth software deployment, bugs or defects rank highly, as their detection delays deployment timelines and increase maintenance costs. The scale of this problem is clear from the huge number of bugs reported daily in open-source software. For example, Microsoft generates about 30,000 bugs each month. Bug triaging or prioritization is essential for managing these defects, allowing them to be classified by severity and urgency.

However, the task of differentiating between critical and less critical defects remains one of the primary challenges in software engineering. Implementing effective prioritization methods not only updates bug management but also assists developers in making good decisions regarding bug fixing techniques. But, the growing number of bugs poses a difficult challenge to ensuring software quality, demanding a closer examination of existing processes and practices. This study assumes the literature review to identify the most effective techniques employed by practitioners and software development experts in prioritizing bugs or defects within the domain of software engineering.

The current methods and strategies in bug hierarchy are:

- Machine Learning Algorithms: Algorithms like Naive Bayes, Decision Trees, and Random Forests are used to predict bug priority. Among these models, Random Forests and Decision Trees resulted in higher accuracy.
- KNN Text Classification: The improved KNN algorithm with clustering reduces complexity and enhances accuracy.
- CNN-Based Prioritization: Utilizes NLP and emotion analysis for better accuracy in bug report prioritization.
- Feature Reduction and Clustering: Combined NMF and PCA for feature reduction, followed by clustering and classification, resulting in improved performance.

And the drawbacks of current methodologies are as:

- Manual Effort: Significant time and resources are required for manual prioritization.
- High Complexity: Traditional KNN algorithms have high computational complexity.
- Limited Features: Some models use limited features, leading to information loss.
- False Positives: Bug-finding tools often yield high false-positive rates.
- Overfitting: Models without feature reduction techniques may overfit and struggle to scale.

B. Literature Survey

Bug prioritization is crucial for the effective bug identification in the software systems, which allows the reduction of the number of bug reports. An individual's handling of bug reports is labor intensive and may be slowed down by other, more vital problems. In this research, the situation has prompted different automated methods that include techniques like NB, Decision Tree and Random Forest [2]. This framework aims to make the priority determined by the items excluded by the bug report. The study conducted an experimental study using Eclipse, an open source application, and the popular browser Firefox, which revealed the viability of the offered method for predicting critical bug reports in [2].

Another work illustrates the KNN text classifier improvements to reduce the computing complexity and data reduction issues [3]. This algorithm is developed based on the medoids that it clustership has the ability to classify samples with less number of samples and higher accuracy in comparison to the previous algorithm [1]. Hence the algorithm implements compression and clustering which decreases the complexity of training while maintaining the value of each laboratory sample. Results of simulations confirm higher accuracy as well as algorithm's ability to lower computational costs [1].

Moreover, in view of this trend, automatic bug report prioritization becomes popular using Convolutional Neural Networks (CNNs) [1]. CNN-based techniques, NLP and emotion algorithms, are applied to the sentiments preprocessing of issue reports and the initialization of the issue importance level [1]. These strategies however, dramatically surpass the anticipated results in the case of modern techniques as they have an ability to improve the correctness as well as accuracy of prioritization of bug reports. The evaluations based on the projects in open sources depict the advantage of the CNN based methods in term of bug report priority prediction [1].

Besides, one of the other research focal areas has been bug prioritization model precision improvement using feature reduction and clustering approaches [4]. These models achieve their outcome by implementing dimensionality reduction, clustering, and classifiers to enhance the correctness of bug severity determination and lower the computational complexity. Experimental outcomes reveal that there is a hopeful gain performance and no doubt, cluster and feature reduction are easier to apply than traditional bug prioritization models [4].

From the above, it can be seen that diverse procedures and approaches are used in the studies which imply the need of automated techniques and recommender systems as a valuable tool for triaging processes optimization. These studies findings are an essential element of better bugs prioritize approaches.

III. PROPOSED APPROACH

We have proposed a framework that combines nature-inspired algorithms such as GA and PSO to optimize bug hierarchy. This framework focuses on identifying relevant features for severity forecasting and employing the feature weighting method. The process comprises four stages, utilizing performance indicators like expertise rating, severity of a bug, and average debugging time to rank programmers. More specifically, classification algorithms such as KNN are utilized to predict bug intensity levels. Experimental evaluations involving PSO-KNN and GA-KNN methods demonstrate the effectiveness of the proposed model. The need to use GA and PSO, and how are they better than others are as:

- Efficient Search: GA and PSO efficiently search large, complex spaces for optimal solutions.
- Adaptability: These algorithms handle non-linear, multi-dimensional problems effectively.
- Improved Performance: GA and PSO optimize parameters and feature sets better than traditional methods, enhancing accuracy and reducing manual effort.

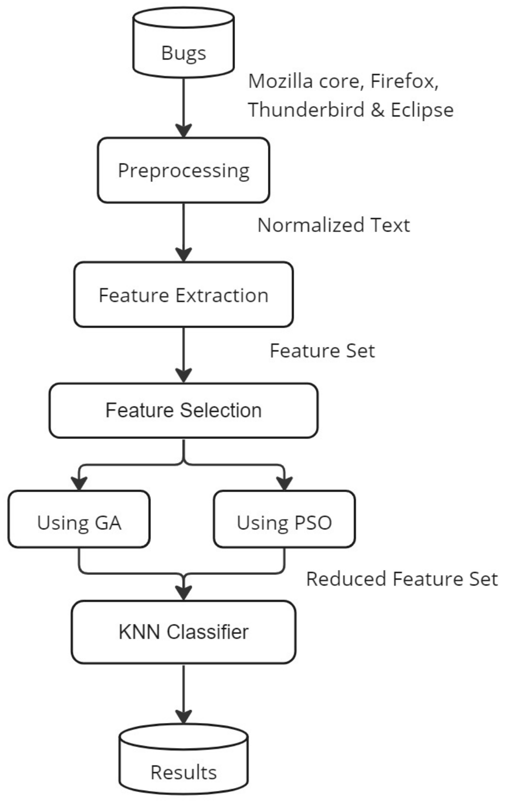

Fig. 1. Flow diagram

A. Dataset

In this study, we collected bug report datasets from four different platforms: Mozilla Core, Firefox, Thunderbird, and Eclipse, using Bugzilla. Each bug report includes fields such as BugID, Severity, Priority, Assignee, Status, Opened, Closed, Resolution, Summary, and Total Days Taken.

TABLE I
BUG REPORT FEATURES

Dataset	Total no. of Features	Reduced Feature Size
Eclipse	8463	4159
Firefox	4587	2254
ThunderBird	4358	2114
Mozilla	5573	2716

B. Preprocessing

1) Tokenization: Tokenization is the process of clustering the whole text into smaller pieces, known as tokens, which are individual words in a document. The objective of tokenization is to preprocess the words before further processing by breaking the text into wordpieces where each word forms its own entity. Accordingly, tokenization is a very powerful part of the classical NLP tasks, such as sentiment analysis and topic modelling.

2) Stop-word removal: Stopwords are words that are frequently utilized in a document, even though they do not make a significant meaning, for example, "the", "a", and "‘an‘." The removal of stopwords is a common software engineering technique employed in text processing in order to boost the accuracy and efficiency of further processes. After removing stopwords, other key words in the text become more important and contribute better to the essentials in the contents of the text.

3) Punctuation marks removal: To generate the processed text and to avoid any errors, deleting punctuation marks is desirable since those are usually unnecessary for text analysis. Punctuation deletion is defined as removing every type of punctuation mark everywhere like commas, periods and semicolons in the text.

4) Stemming: Stemming means that stems of the words are cut off. This method is used to visually omit words that are different only by the sequence so "run", "runnin" and "ran" will all be converted to "run". Through elimination of this type of words, stemming narrows the vocabulary and boosts the performance of text analysis.

5) TF-IDF score: This is the main way of converting the words that are available in a feature vector, into numeric features. TF-IDF, which has been used frequently in information retrieval, computes how common and how unique it is among the whole collection of documents (IDF). The TF and IDF score is calculated for each term in the festival. And the TF-IDF score is the product of the TF*IDF score. This algorithm is more than just the word's score as it calculates its relevance across all bugs in the dataset. For instance, the weight of a word v in a bug report b is determined as:

$$W_{w,d} = \mathrm{TF}_{w,d} \times \log\left(\frac{n}{\mathrm{DF}_w}\right)$$

where n represents the total number of documents in the dataset, $\mathrm{TF}_{w,d}$ denotes the occurrences of word w in document d, DF_w signifies the number of documents containing word w.

6) Feature Selection: The dataset was obtained by normalizing the texts from pre-processing, after which feature extraction was followed to make a feature set. Feature extraction consists of the selection of relevant data from textual bug reports, for example, the bug description, bug severity, the developer's name assigned to the bug, and other related metadata. The objective of the feature extraction is to simplify the data to the bare essentials while at the same time maintaining the most relevant of the precious details. Fitted features are then used in the construction of a feature set, which is simply a structured description of the data and is suitable for further analysis. Features extracted would be transformed into a feature set, and this feature set can consequently be used for further analysis. The hallmark of a supervised learning algorithm usually is a group of the labeled cases that each case comprises of the features and a class that represents the class or category to which this case belongs. Then, following the feature set, it's used to champion machine learning models for bug triage that can accurately sort new bug reports based on their features.

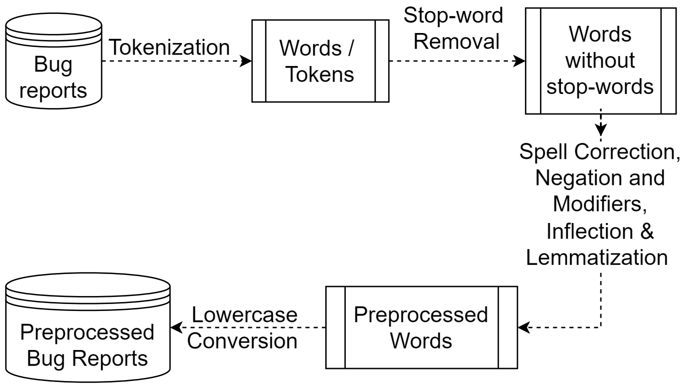

Fig. 2. Flow diagram

C. Classification-Based Approach

The k-Nearest Neighbors is a non-parametric classification technique that is known for its simplicity and effectiveness. To classify a data record q, the method identifies its k nearest neighbors, forming a neighborhood around q. A decision on the classification of q (i.e. the label of q) depends on weighted averaging of the class-labels (voting) in the neighbourhood. Some methods use a correction that takes distances into account. Yet, for k-nearest neighbor application, choosing the k-value that is the proper one is a prerequisite and our classification is tightly dependent on this value. There is a myriad of options for choosing the k value. Of the many possibilities, a simple one is as follows: run numerous iterations with different k values and pick the best.

Having kNN be no more affected by the choice of k, a solution is to look at different ranges of k-nearest neighbours instead of just the neighbours that are closest to k-nearest. We propose an algorithm that aims to give a more reliable support value to better reveal the true class of q. It does so by aggregating the support of multiple sets of nearest neighbors for various classes. This idea is based on contextual probability. It is, however, less controlled by k, and can achieve similar classification performance for optimal value of k.

D. About PSO and GA

1) PSO: Particle Swarm Optimization(PSO) is a computational optimization algorithm that is inspired by the social behaviour of birds and fishes. It is a kind of meta-heuristic (swarm modeling) and population-based optimization approach which heuristically solves a problem by recurring search of a population of candidate solutions.

In PSO, a group of particles is used to represent the candidate solutions of a problem each moving through the search space. Every particle's movement is directed by its own engagement and the engagement of its neighbors. The new position and velocity of particles in the population are calculated as a function of their current position, velocity, and the current best solution from itself and its neighbors.

The following steps are used as a solution approach of PSO for bug prioritization:
1. Define the problem: The foremost requirement is to address the task's problem statement, for example, bug prioritization. objective, constraints, and decision variables. The objective function will imply the cruciality of bugs depending on some parameters for instance the degree, period and effect of the bug on the system.
2. Represent the bugs as particles: Each bug can be depicted as a particle, and the particle coordinates will designate it. The bug's criticality is reflected by the action. The trajectory of the particle can be visualized using the particle's position which is a vector of real numbers as shown in fig. 3.
3. Initialize the particles: randomly created with a number of particles which constitute the pool. The objective function is used to evaluate the fitness.
4. Set the velocity and acceleration: We will assign some random values to the particles
5. Update the particle's position and velocity: The information of the particle will be update again and again after every iterations.
The values will be updated using following equations:

$$\begin{aligned} \text{vel\_new} = \omega \cdot \text{vel\_old} \\ + \alpha \cdot \beta_1 \cdot (\text{pos\_pbest} - \text{pos\_current}) \\ + \gamma \cdot \beta_2 \cdot (\text{pos\_gbest} - \text{pos\_current}) \end{aligned} \tag{1}$$

$$\text{pos\_new} = \text{pos\_current} + \text{vel\_new} \tag{2}$$

where β_1 and β_2 are Stochastic values, ω is the Inertia coefficient, α and γ are the Acceleration constants ,pos_pbest is the Optimal individual position, and pos_gbest is the Swarm's optimal position.

6. Evaluate the particle position: In this the particle's performance are evaluated on the basis of the objective function.
7. Update the Optimal individual position: On the basis of better fitness value of a particle generated by the objective function the optimal individual position is changed.
8. Update the Swarm's optimal position: Now on the basis of better Optimal individual position the current swarm's optimal position is replaced.
9. Repeat steps 5-8: Steps 5-8 will be carried out over and over again until the termination criteria are fulfilled as shown in fig. 3. Stopping criteria can be the, by limiting iteration to a range, certain threshold fitness value, or minimum changeover
10. Select the best solution: The Swarm's optimal position represents the optimal solution, which provides the most critical bugs that need to be fixed first.

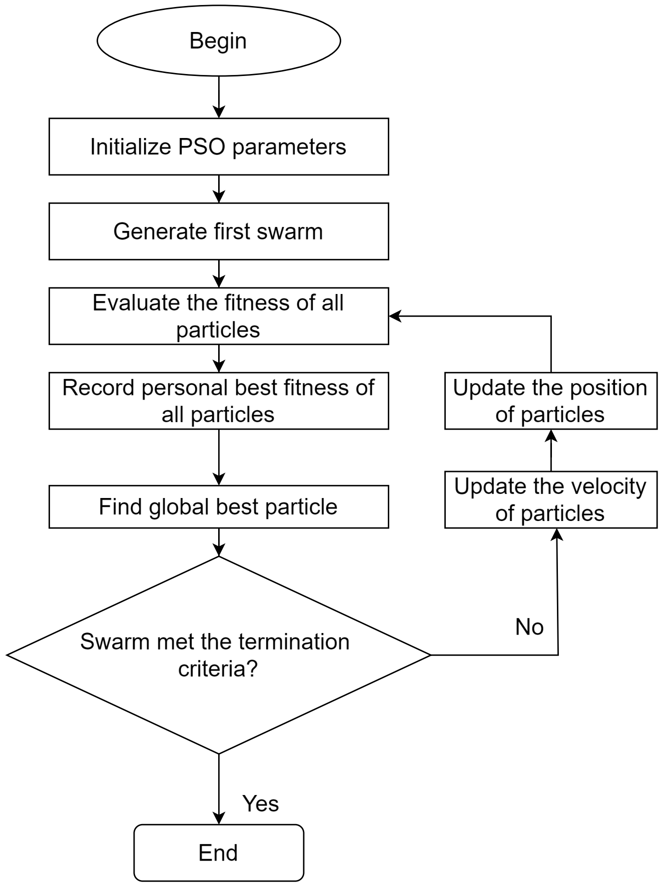
Begin
Initialize PSO parameters
Generate first swarm
Evaluate the fitness of all particles
Record personal best fitness of all particles
Find global best particle
Swarm met the termination criteria?
No
Update the velocity of particles
Update the position of particles
Yes
End

Fig. 3. Flow diagram

2) GA: Genetic Algorithm (GA) operates based on the process of natural selection and genetics. For this to work, GA relies on the "population of candidate solutions", iteratively updating them by a way of traditional operators.
The main operators are:
1. Selection: In this the operator choose the best candidates for the population to become the parents.
2. Crossover: After the Selection operation, the selected two candidates provides a child after mixing their genes.
3. Mutation: The new born child(gene) now provided some of the random features so that it works better than their parents.
And this operations runs till a limiting condition
The following steps are used as a solution approach of genetic algorithm for bug prioritization:
1. Define the problem: A first step toward defining

the bug prioritization problem is to state the problem, including the stakeholders, constraints, and decision variables. The objective function is the criterion that is used to measure the assessment of bugs subject to a variety of factors, namely, the severity, repetition and lack of functionality, amongst others.
2. Represent the bugs as chromosomes: Every bug can be encoded as a gene, where each gene represents one chromosome. The chromosomes can be encoded using binary or real-valued representation.
3. Initialize the population: A group of chromosomes is generated randomly and each chromosome is examined based on the objective function.
4. Selection: The process of selection involves the selection of most suitable chromosomes in a population. taking into account their fitness score. Fitness value is derived from the objective function.
5. Crossover: The crossover operator works by means of selection of two parent chromosomes and further creation. The crossover operator participates in the process of crossover facilitation finding new solutions which might be better than the original.
6. Mutation: A mutation operator involves randomly swapping a gene in a chromosome. The mutation operator basically allow the creation of new genetic diversity in the population.
7. Evaluate the offspring: The productivity of the offspring chromosomes is calculated using the mathematical fitness value.
8. Replace the population: The offspringd are inserted into the population, and the least capable chromosomes are removed. This is an effective way to preserve the diversity in environment, delaying premature convergence.
9. Repeat steps 4-8: Steps 4-8 carry on being repeated until the stopping criteria is satisfied. Stopping criteria can be the, by limiting iteration to a range, certain threshold fitness value, or the minimum improvement in evolution process.
10. Select the best solution: The final population has the fittest chromosomes, which were selected.

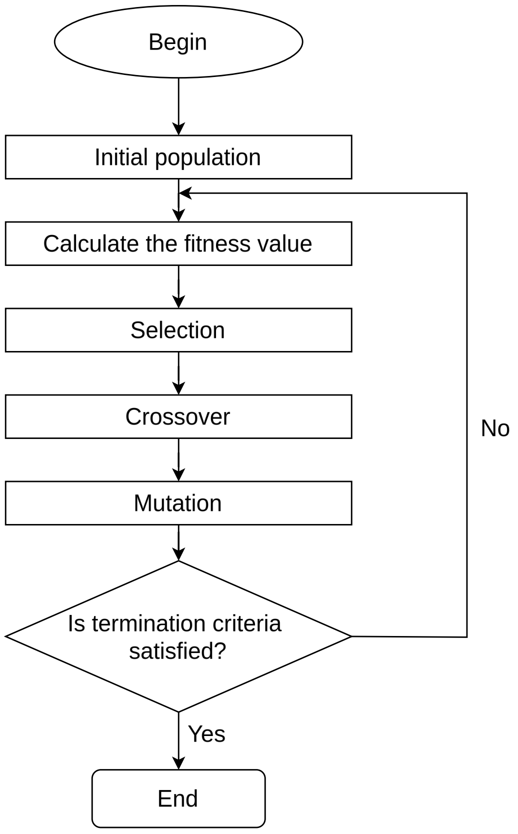
Begin
Initial population
Calculate the fitness value
Selection
Crossover
Mutation
Is termination criteria satisfied?
No
Yes
End

Fig. 4. Flow diagram

IV. RESULT

Python programming was utilized for the improvement and execution of the bug prioritization strategy. The system was assessed on four bug report datasets (Thunderbird, Firefox, Eclipse, and Mozilla) in order to survey their viability. Severity of bugs were determined using various parameters including precision, recall, accuracy, and f-measure. A study was conducted to evaluate the impact and severity of the bugs, so that a programmer can prioritize the bugs for fixing according to their severity.

The study conducted bug severity running simulations on four popular bug datasets. The results indicate that the GA-KNN is good than PSO-KNN method on the basis of average accuracy. On the other hand, PSO-KNN was found to provide more accurate bug severity predictions than KNN techniques. The proposed PSO- based and GA-KNN feature selection method significantly improves the results of KNN computations. The results of the proposed algorithms for assigning the bugs, including PSO-KNN and GA-KNN approaches, using the four datasets are shown in the Table-II:

The proposed PSO-KNN and GA-KNN seemed to work even on quite large datasets, while the other traditional algorithms failed to do so. For example, PSO-KNN and GA-KNN worked pretty well even after taking 5000 columns of the dataset into consideration. On the other hand, other algorithms didn't work as expected when increasing the number of columns.

The results of PSO-KNN and GA-KNN are then compared with traditional KNN and other methods, shown in Figure 13.

A. *GRAPHICAL REPRESENTATION OF GA AND PSO*

MOZILLA PLATFORM

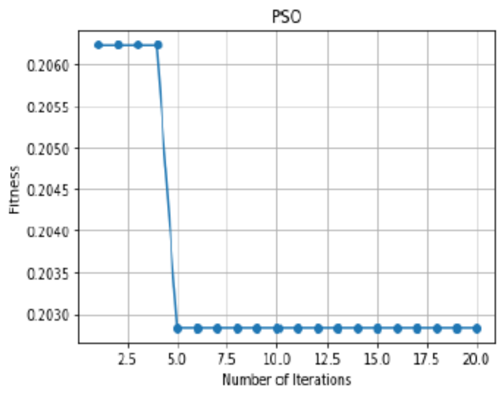

Fig. 5. PSO Algorithm

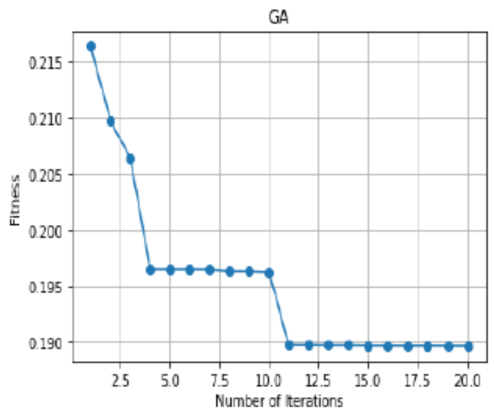

Fig. 6. Genetic Algorithm

ECLIPSE PLATFORM

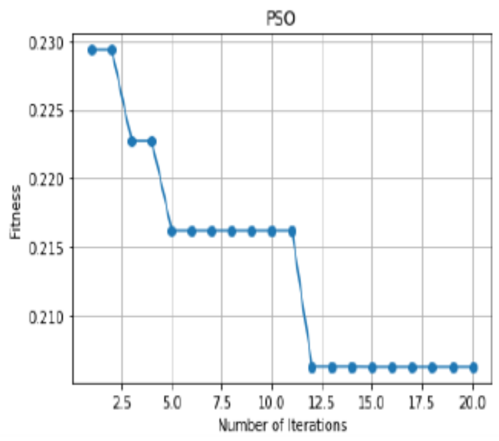

Fig. 7. PSO Algorithm

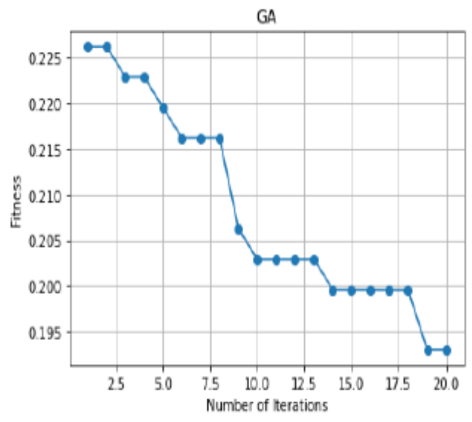

Fig. 8. Genetic Algorithm

B. *CONFUSION MATRIX OF GA AND PSO*

MOZILLA PLATFORM

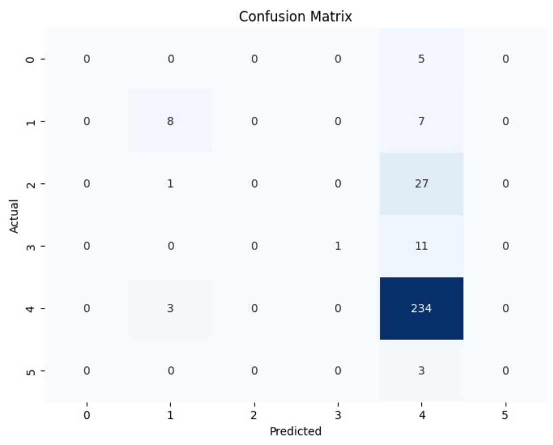

Fig. 9. PSO Algorithm

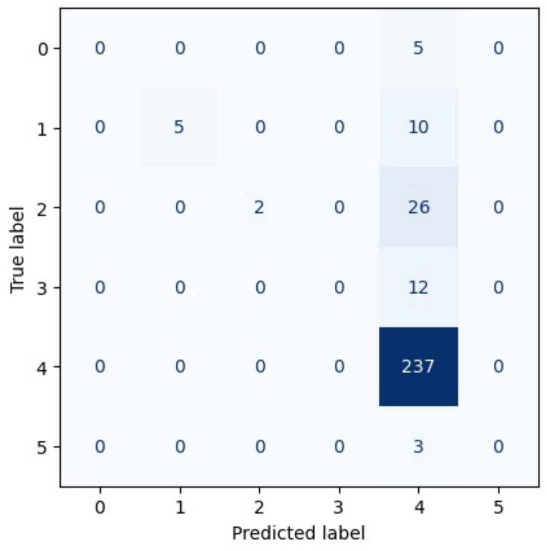

Fig. 10. Genetic Algorithm

ECLIPSE PLATFORM

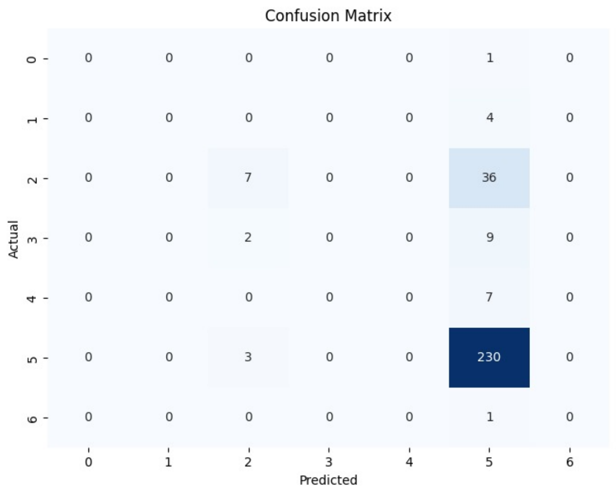

Fig. 11. PSO Algorithm

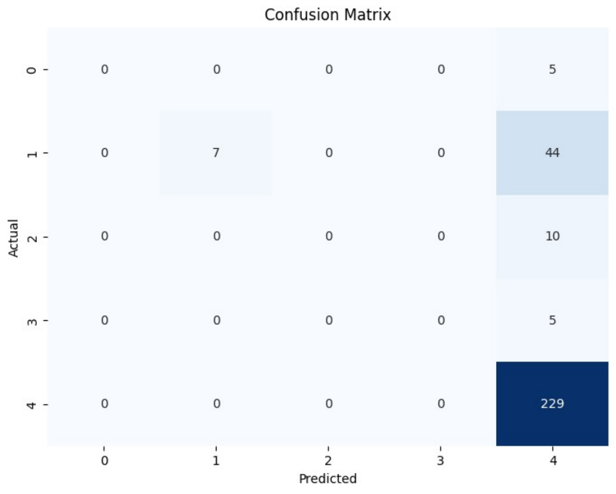

Fig. 12. Genetic Algorithm

V. DISCUSSION

Many machine learning algorithms have been used to predict the priority levels of bugs. Among all the known methods available in the various researches, we used SVM, LR, Decision Trees, NB, Random Forest, CNN, GA, and PSO. Every one of these methods has unique qualities, benefits, and drawbacks.

SVM has been utilized widely in the bug prioritization models due to its ability to deal with multi-dimensional information well and its capacity to differentiate between the complicated choice boundaries. Decision Tree works well for investigatory examination and decision-making strategies since they are basic to implement.

On the other hand, NB is based on the assumption of autonomy among highlights and is especially viable for content classification assignments, such as bug report prioritization. Random Forest, an outfit learning strategy, combines numerous choice trees to move forward accuracy and decrease overfitting. CNN, essentially utilized for image classification, has been adjusted for bug report prioritization by changing over printed data into vectors and analyzing the emotional values related with bug reports.

Both GA and PSO are meta-heuristic optimization algorithms that give us an approach to handling complex optimization issues. While GA mimics the forms of characteristic choice and advancement to discover ideal arrangements, PSO simulates the behavior of cluster of particles to effectively investigate the look-space.

When comparing these techniques, SVM and LR are preferred due to their interpretability and ease of implementation. NB is appropriate for text-based bug prioritization operations due to its simplicity and effectiveness. Random Forests combine the qualities of Decision trees while controlling their shortcomings. CNNs exceed expectations at capturing complex patterns in text data.

However, GA and PSO stand out for their ability to handle complex optimization issues and look for spaces to be optimized. They offer a narrow approach to bug prioritization by optimizing the included choices and demonstrating parameters at the same time. Besides, GA and PSO can adjust to all of the situations and the ever-improving datasets, which makes them appropriate for real-world bug prioritization scenarios.

In conclusion, traditional machine learning calculations like SVM, LR, Decision Trees, NB, Random Forests, and CNN offer some good knowledge and execution in bug prioritization, while GA and PSO give supporting qualities in optimization and flexibility, making them promising algorithms for improving bug prioritization models as shown in Table-II and Table-III.

VI. CONCLUSION

Our bug prioritization system for bug triage employed in meta-heuristic feature selection algorithms aims at optimizing the quality of software development projects and improving the organization efficiency. This system will automate the bug triage and thus it saves time and effort needed for bug resolution.

The approach utilizes a number of techniques like NLP-based data preprocessing, feature extraction, and feature selection using PSO and GA algorithms to come up with a feature set that is less dimensional but informative and relevant. Then it classifies the bug report into different categories based on its feature and attribute with the help of machine learning algorithms.

Overall, the bug prioritization model for bug triage with meta-heuristic feature selection algorithms for optimization is a powerful tool that can enhance the efficiency and quality of software development ventures. With more improvement and polishing, this system can be vital for the companies which will be thinking to

TABLE II
RESULTS

Approaches	Dataset	Performance			
		Accuracy	Precision	Recall	F-measure
K-neighbour	Mozilla	79.33	66.9	79.3	71.5
	Firefox	57.66	54.80	57.70	49.40
	Thunderbird	63.26	54.90	61.70	57.30
	Eclipse	72.66	65.30	72.70	68.60
	Avg. results	68.75	60.47	67.77	61.20
PSO K-neighbour	Mozilla	81.33	78.2	81.30	74.40
	Firefox	66.66	59.80	66.70	57.00
	Thunderbird	69.00	64.80	69.00	58.70
	Eclipse	79.66	72.10	79.70	74.00
	Avg. results	74.16	72.40	74.18	66.00
GA K-neighbour	Mozilla	82.33	82.90	82.30	75.80
	Firefox	70.67	69.80	70.7	63.4
	Thunderbird	72.46	63.50	71.30	61.80
	Eclipse	80.66	76.50	80.70	74.00
	Avg. results	76.52	73.17	76.25	68.75

TABLE III
RESULTS

Approaches	Dataset	Performance			
		Accuracy	Precision	Recall	F-measure
SVM	Mozilla	77.60	30.05	19.31	20.23
	Firefox	73.75	34.76	20.29	19.96
	Thunderbird	74.66	30.75	21.85	22.37
	Eclipse	73.86	10.55	14.28	12.13
	Avg. results	74.96	26.52	18.93	18.67
Random Forest	Mozilla	76.93	36.50	20.05	21.10
	Firefox	68.5	28.30	25.72	24.55
	Thunderbird	74.91	39.10	25.61	27.03
	Eclipse	67.88	24.51	16.13	15.47
	Avg. results	72.05	32.10	24.60	22.03
Decision Tree	Mozilla	69.27	18.97	17.84	18.19
	Firefox	54.50	25.74	25.27	25.42
	Thunderbird	67.17	33.23	29.67	31.07
	Eclipse	58.39	19.40	18.03	18.41
	Avg. results	62.33	24.08	22.07	23.27
Logistic Regression	Mozilla	77.2	21.16	16.80	16.52
	Firefox	66.66	23.02	18.96	17.28
	Thunderbird	74.33	31.61	19.72	19.52
	Eclipse	69.58	22.84	16.00	14.83
	Avg. results	71.94	24.65	17.87	17.03

accelerate their software development and make the quality products for their customers.

Additionally, it is also possible to see that Genetic algorithm and Particle Swarm Optimization works better than SVM, Decision Tree, Random Forest in terms of operational efficiency.

VII. FUTURE SCOPE

There are several promising future research directions and potential advancements for using meta-heuristic feature selection techniques such as PSO and GA for bug triage. Here are a few:

- **Reducing Computational Complexity:** A further research can be oriented to the development of more refined algorithms and the invention of hybrid techniques, which involve the association of heuristic methods with the other methods. Such association should be allowed to limit the number of iterations necessary for the channels, thus leading to increase in the feasibility to large software projects.
- **Improving Feature Selection Quality:** Integrating new techniques for further feature processing along with using domain knowledge can result in finding more feature representation and hence an ability to achieve a convergence better than others.
- **Mitigating Over-fitting:** Future research can focus on the methods for preventing over-fitting, for example, regularization approaches, cross-validation, and using more robust validation databases to make the output of the system not only acceptable for the current data, but also more general for the next.

VIII. ACKNOWLEDGEMENT

Our sincere gratitude goes to Dr. Ajay Nehra for his contributions and the feedback he provided to us, due to which the quality of this paper improved drastically. He has been a constant guidance and thoughtful reviewer throughout the whole process, which has served us greatly in honing

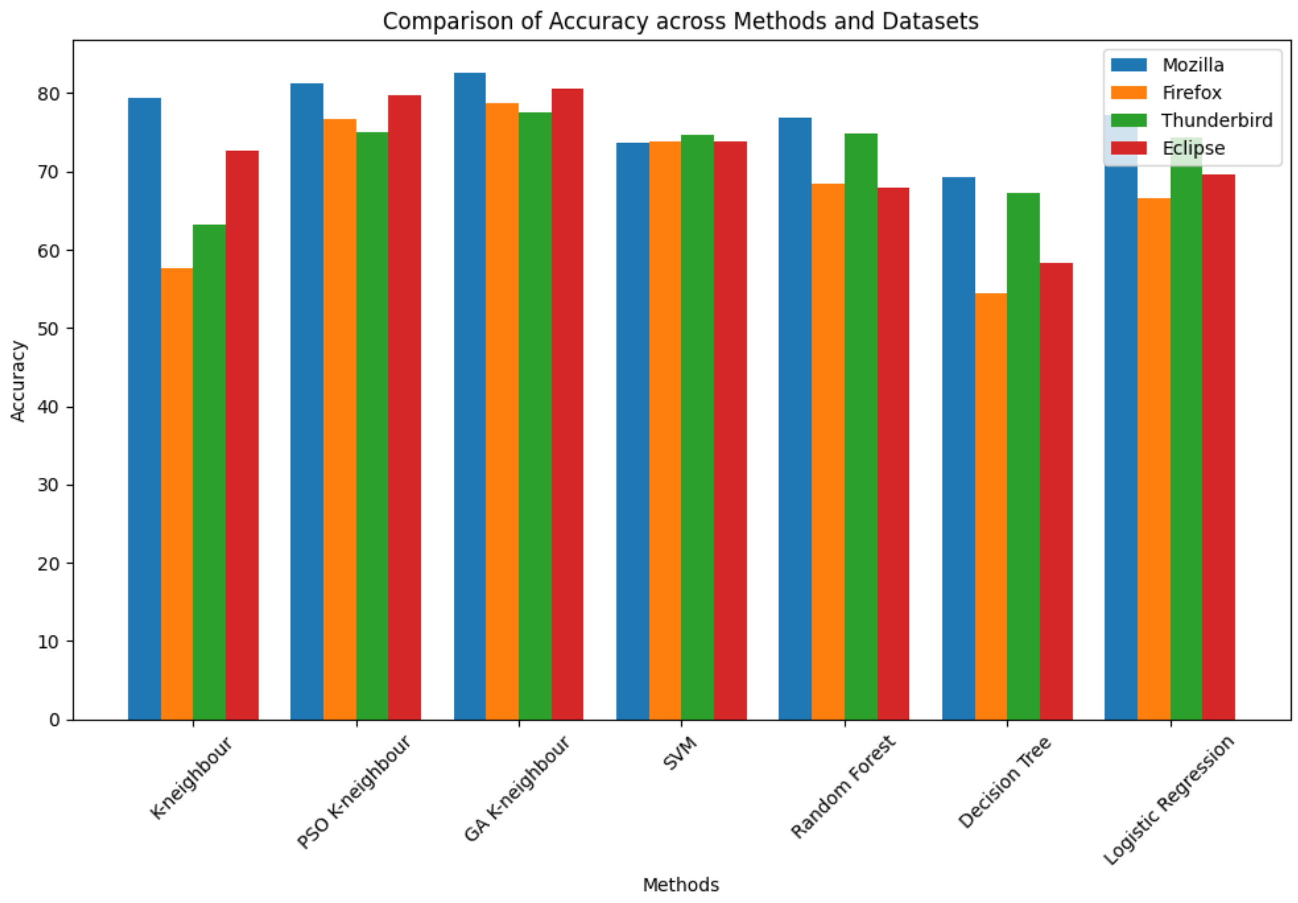

Fig. 13. Results

our work. Further, we are thankful to the Department of Computer Science & Engineering, IIIT Kota for providing us the required platform to take up this project.

References

[1] Qasim Umer, Hui Liu, & Inam Illahi (2023). *"CNN-Based Automatic Prioritization of Bug Reports"* [Online].

[2] Mamdouh Alenezi & Shadi Banitaan (2013). *"Bug Reports Prioritization: Which Features and Classifier to Use?"* [Online].

[3] Yong, Zhou, Youwen Li, & Shixiong Xia (2009). *"An Improved KNN Text Classification Algorithm Based on Clustering"* [Online].

[4] Shahid Iqbal, Rashid Naseem, Salman Jan, Sami Alshmrany, Muhammad Yasar & Arshad Ali (2020). *"Determining Bug Prioritization Using Feature Reduction and Clustering With Classification"* [Online].

[5] Mudita Juneja & S.K. Nagar (2016). *"Particle Swarm Optimization algorithm and its parameters: A Review"* [Online].

[6] Tom V. Mathew. *"Genetic Algorithm"* [Online].

[7] Ruchika Malhotra, Ajay Dabas, Hariharasudhan A S, Manish Pant (2021). *"A Study on Machine Learning Applied to Software Bug Priority Prediction"* [Online].

[8] Bugzilla [Online]. Available: https://www.bugzilla.org/

[9] Wikipedia [Online].
Available: https://en.wikipedia.org/wiki/Genetic_algorithm

[10] Wikipedia [Online].
Available: https://en.wikipedia.org/wiki/Particle_swarm_optimization