

Received September 27, 2020, accepted October 15, 2020, date of publication November 10, 2020, date of current version December 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3035063

Determining Bug Prioritization Using Feature Reduction and Clustering With Classification

SHAHID IQBAL¹, RASHID NASEEM², SALMAN JAN³, SAMI ALSHMRANY⁴, MUHAMMAD YASAR⁵, AND ARSHAD ALI⁴

¹Department of Computer Science, City University of Science and Information Technology, Peshawar 25000, Pakistan

²Department of IT and Computer Science, Pak-Austria Fachhochschule: Institute of Applied Sciences and Technology, Haripur 22600, Pakistan

³Department of Computer Science, University of Peshawar, Peshawar 25120, Pakistan

⁴Faculty of Computer and Information Systems, Islamic University of Madinah, Medina 42351, Saudi Arabia

⁵Malaysian Institute of Information Technology, University of Kuala Lumpur, Kuala Lumpur 50250, Malaysia

Corresponding author: Salman Jan (salman@uop.edu.pk)


ABSTRACT Assigning accurate and timely priorities to bugs manually is resource consuming and effects addressing important bugs. In the existing work single feature is used which leads to information loss because bugs have a lot of features including “severity”, “component”, “operation system”, “owner”, “status”, “assigned to”, “summary” etc. In this research, the authors proposed an improved model based on problem title, severity, and component for bug prioritization. We converted these textual features to numeric features using Term Frequency Inverse Document Frequency. During conversion, 5591 new features are generated, which increase complexity and running time of algorithms. To minimize these aspects, non-negative Matrix Factorization (NMF) and Principal Component Analysis (PCA) algorithms are used. Our proposed model is a combination of feature reduction, clustering, and classification algorithms. Clustering is performed on all and reduced features. For clustering X-Mean and K-Mean algorithms are used. SVM and Naive Bayes classifiers are applied on all features, reduced features, and on clustered features. For experiments chromium, eclipse, net beans, mozilla, and free desktop datasets are used. Experimental results reveal better performance of model, both with all features and with reduced features in terms of precision, recall, f-score, and accuracy. Maximum improvement is achieved with reduced features. With all features chromium, eclipse, free desktop, mozilla and net beans achieved 22.46%, 8.32%, 30.93%, 25.79% and 37.78% respectively improvement in accuracy. With reduced features chromium, elipse, free desktop, mozilla, net beans achieved 14.64%, 8.81%, 33.22%, 34.37% and 41.01% accuracy respectively. Overall classification with clustering and reduced features performed better than classification on all features, classification with clustering on all features, and classification on reduced features. In all the approaches SVM classifier outperformed Naive Bayes in terms of precision, recall, f-score, and accuracy. On average maximum accuracy is achieved by SVM with NMF and X-Mean clustering.

INDEX TERMS Quality software, bugs, textual features.

I. INTRODUCTION

Software systems are becoming necessary for every business. Many organizations depend on software to deal with their day-to-day operations and deliver services to their clients. Thus, the increasing demand for quality software also increases software maintenance costs. According to [1] approximately 90% of software life-cycle cost is consumed by software maintenance activities. With maintenance, software testing is also performed to check the quality of

software. In software testing, one of the most important activities is Bug Triaging. Critical decisions can be taken in bugs fixing with the help of bug triaging. It also helps in finding duplicate reported bugs, correct and incorrect reported bugs, bugs that require immediate attention, and which does not, and assigning an appropriate developer to it. However, manual bug triaging is a tedious, time, and resource-consuming process. This problem is addressed by different researchers in [2]–[6] and Uddin *et al.* [7]. According to Guo *et al.* [8] and Uddin *et al.* [7] up to August 2009, the Eclipse bug database contains over 250,000 and the Mozilla bug database over 500,000 bug reports. On average, Eclipse received 120 and

The associate editor coordinating the review of this manuscript and approving it for publication was Mamoun Alazab .

Mozilla 170 new bug reports on each day from January to July 2009. So, it is a very time consuming and tedious job for bug triager to triage these bugs daily. In a normal bug triaging process, when a new bug enters the bug repository, the triager analyzes the bug report in two ways for taking decisions i.e. Repository Oriented Decisions (ROD) and Development Oriented Decisions (DOD). In ROD, once it is verified that the reported bug is not duplicate its validity is checked i.e. it is Correct or Incorrect. This verification and validation help discard the bug reports which do not need to be resolved. In DOD, the triager assigns Severity and Priority to bug reports. So that critical bugs should be fixed on priority bases [2], [7]. In the end, the triager writes his remarks regarding the reported bug, and the bug is assigned to the appropriate developer for resolution. Hence an automated system is required to assign accurate and timely priority to newly reported bugs. For achieving this task researchers have used different data mining techniques like clustering, classification, etc. But still, improvement is required because assigning correct priority to reported bugs plays an important role in fixing critical bugs on time. An improved model has been proposed in this work to achieve this task. According to [6], [9], [10] when non-supervised machine learning algorithms are applied as preprocessing steps like Clustering, before applying supervised machine learning algorithms like Classification on data, it improves results. This approach is the first time used by Goyal *et al.* [6].

A. RESEARCH SIGNIFICANCE

The proposed model has significant effects on bug prioritization. It has enhanced the accuracy of bug prediction and is helpful for open source software (Chromium, Eclipse, NetBeans, Mozilla, and FreeDesktop) for fixing their critical bugs on a priority basis. The proposed model can be implemented in software houses for the improvement of their software quality.

The remainder of this research article is organized as follows. Section II provides an overview of bug prioritization and literature review. This chapter discusses the different techniques used by researchers. The review of comparative published literature and evaluation criteria is also discussed. Section III presents the research methodology. The first phase presents the steps for analysis of the existing bug prioritization model, finding problems in the existing model, and propose improvements. Second phase incorporating proposed improvements in the existing system. Section IV presents and discusses the experimental results of the proposed system and compare results with existing approaches. Section V concludes this papers with accomplished goals and contributions.

II. LITERATURE

Different Researchers have worked on bug reports, detecting duplicate bugs, assigning severity, priorities, and developer to a bug. There is a number of studies in this field, different bug-finding tools are developed to find bugs from source code

and prioritize them, but the usually high false-positive rate is observed in prioritization. A lot of researchers proposed different ways of bug prioritization improvement.

A. BUG FINDING TOOLS

Kim and Ernst [11] applied three bug-finding tools FindBugs, JLint, and PMD on open source projects Columba Lucene, and Scarab to assign priority levels and analyzing the lifetime of a bug. Software change history (history-based data) is used for analyzing bug lifetime and a higher priority is assigned to bug with a shorter life. Kim and Ernst [12] also worked on the weight of bug categories. The weight of a bug category depends on the resolution of a bug in that category. When a bug is resolved in a bug category, that category weight is increased. Kremenek and Engler [13] also used bug-finding tools for prioritization based on a frequency count of successful and failed checks. Based on the tool's analysis decisions successful and failed checks are classified.

B. MACHINE LEARNING ALGORITHMS

Anvik *et al.* [14] applied machine learning algorithms on bug reports for assigning developers automatically to the newly reported bug. Decision Trees (DT), Naive Bayes (NB), and SVM were used on Firefox, Eclipse, and GCC open-source projects. Among DT, NB, and SVM, high precision was achieved by SVM. Anvik and Murphy [15] compared their approaches by retrieving the expertise of developers from source code repositories and bug repositories. They found that both approaches are good at assigning appropriate developers for bug fixing. Anvik and Murphy [2] proposed a model for building recommenders for the variation of DODs. e.g. finding people interested in a bug, predicting component of newly reported bug, and assigning a developer to the new bug. GCC, Bugzilla, Eclipse, Mylyn, and Firefox open-source projects are used for the evaluation of recommenders and achieved more than 70% precision. For the recommender's configuration, an automatic approach is proposed which reduced configuration efforts. Researchers have used different classification techniques for predicting suitable developers. In the proposed work of Murphy and Cubranic [16] supervised Bayesian learning approach is used, which can accurately assign 30% of the reports to developers. Anvik *et al.* [14] used a supervised ML algorithm and presented a semi-automatic approach for assigning bug reports to a developer with his appropriate expertise for bug resolution. Ahsan *et al.* [17] used SVM, Naive Bayes, Decision Tree, Random Forest, Reduces Error Pruning (REP) Tree, Tree-J48, and Radial Basis Function (RBF) Network and presented comparative analysis in order to automatically assign bug report to a developer in an optimized way. Lamkanfi *et al.* [18] analyzed the textual description of bug reports by using text mining algorithms, for predicting the severity of bug reports accurately. Studies [3] extended their previous work by using SVM, K-Nearest Neighbour (KNN), NB, and NB Multinomial Classifiers, and found that NB

Multinomial Classifier results best than other proposed algorithm. Chaturvedi and Singh [19] applied SVM, KNN, NB, NB Multinomial, J48, and RIPPER for assigning Severity to reported bug. For assigning bug priority Yao *et al.* [20] used Artificial Neural Network (ANN) and for increasing the accuracy of the model several strategies are proposed. Kanwal and Maqbool *et al.* [4] used NB and SVM classifiers for bug prioritization and found SVM perform better than NB. Also, two new measures (Nearest False Negatives and Nearest False Positives) are proposed by them. Sharma *et al.* [21] used Neural Network, KNN, NB, and SVM for predicting the priority of new bugs and evaluated their performance on different measures. Pre-processing step including Tokenization and stop word removal applied to a Summary attribute of bug reports. The accuracy of all techniques was better except for NB. Nigam *et al.* [5] used Inverse Multiquadric, Sigmoid, Radial Basis, Power and Multiquadric Kernel functions with Multi-Class SVM for grouping similar bug reports on Labeled, Unlabeled, and Test data. Experimental results showed that Radial Basis, Multiquadric, and Inverse Multiquadric provided good accuracy. Javed *et al.* [22] proposed an automated bug classification model. For feature selection, they used Chi-Square and TFIDF and for assigning the correct class to reported bugs they used a multinomial Naive Bayes classifier. Kaur and Jindal [23] performed prediction bug severity of thirteen different apache projects, which were automatically extracted from the Bug Report Collection System tool. They predicted severity on the most frequent terms used in the bug report summary attribute. Initially, they used preprocessing on summary and then applied different machine learning techniques including NB, DT, SVM, RF, KNN, Bagging, Boosting, SLDA, MAXENT, and Glnet. They found that the Boosting technique outperformed other machine learning techniques. Kukkar *et al.* [24] proposed a novel deep learning model for multiclass severity classification, using Convolution Neural Network and Random forest with Boosting (BCR). For preprocessing they used the natural language technique on bug report and then n-gram is used for feature extraction. After that CNN extracts important feature patterns of respective severity classes and at the end random forest with boosting classifies the multiple bug severity classes. The average accuracy of the proposed model is 96.34%. Gomes *et al.* [25] conducted a comprehensive mapping study review of the latest research efforts on automatically bug report severity prediction. They categorize their study into ten different quantitative aspects of experiments reported in different papers. Initially, they selected 50 papers and then filter them to 18 papers and adding more 9 papers to conduct a mapping study review. There gathered data confirms the relevance of the topic, reflects the scientific maturity of the research area, as well as identify gaps, which can motivate new research initiatives. A comprehensive summary of different researcher work is given below in Table 1 while Bugs Report example of each dataset is presented in Table 2.

C. INTERACTIONS WITH A BUG REPORT

People play different roles as they interact with reports in a bug repository. The person who submits the report is known as a reporter or the submitter of the report. The triager is the person who decides if the report is meaningful and who assigns the responsibility of the report to a developer. The one that resolves the report is the resolver. A person that contributes a fix for a bug is called a contributor. A contributor may also contribute comments about how to resolve a bug or additional information that leads to the resolution of a report. A person may assume any one of these roles at any time. For example, a triager may resolve a report as a duplicate of an existing report. Alternatively, a developer may submit a report, assign it to himself, contribute a fix, and then resolve the report. For that report, a single person has fulfilled all the roles [14].

D. PROBLEM BACKGROUND

Software development organizations use their significant portion of resources in handling user-submitted bug reports. In the overall life cycle of the software product, 70% of the cost is consumed by maintenance [29]. The software which is commonly used by society, the number of reported bugs typically exceeds the resources available to triage them. As a result, important bugs are entertained very late. According to Anvik *et al.* [14], 3426 bug reports were submitted for Eclipse over the four-month period, averaging 29 reports per day. Manually assigning priorities to this bug is a time and resource-consuming process. Automated bug prioritization will reduce the time and effort of resources. In the bug tracking system, the reported bug severity is assigned by the user, and priority is assigned by the developer. One serious issue in the submitted report is the assigning of accurate severity. In most of the cases, the user is unaware of the difference between categories (Herraiz *et al.* 2008). Developer assigns priorities and allocates their time and resources accordingly. Accurate prioritization helps in bug fixing schedule and resource allocation, otherwise important bugs resolution will be delayed [21].

E. PROBLEM STATEMENT

Manual bug triaging is a very tedious, time and resource-consuming process, due to which important bugs are entertained very late [6]. In the existing model single feature is used for bug prioritization, which leads to information loss, because bugs have a lot of features like “severity”, “component”, “operation system”, “owner”, “status”, “assigned to” and “summary”. To improve the result, we will use three features for bug prioritization i.e, problem title, severity, and component. These are text and categorical features and will be transformed into numeric features because clustering and classification algorithms accept numeric data as input. When converting text and categorical features to numeric features, the total number of features increases to 5591, and these features will increase more if a large number of bug’s dataset

TABLE 1. Summary of approaches used by different researchers.

Reference	Features	Findings	Pros	Cons
[20]	Categorical	ANN model accuracy is better than the NB model for predicting bug priorities.	Different approaches are proposed to improve the accuracy.	Generalization ability may be reduced by Overtraining
[18]	Textual	If training data is provided in a large size, Severity can be predicted with improved accuracy.	Severity can be predicted on description attribute in a bug report, which contains information related to the bug.	For other Projects, the proposed approach is not guaranteed because the tools might contain errors while processing data.
[3]	Textual	Naive Bayes Multinomial performs better than KNN, Naive Bayes and SVM	The more efficient and automated model can be applied for bug triaging process.	For other Projects, the proposed approach is not guaranteed because the tools might contain errors while processing data.
[19]	Textual	Experimental results show the best F-score with the increase in a number of terms, the performance of the proposed technique improves.	The severity level can be predicted of new bug reports, with the help of which triager can assign bug reports to the developer.	For most of the projects, the performance of NB classifier in terms of accuracy did not improve.
[4]	Categorical and textual	For textual features, SVM performs better than Naive Bayes, while for categorical features NB results better than Support Vector Machine.	Two new measures are proposed NFP and NFN, which are found useful for evaluating the performance of classifiers.	Naive Bayes does not perform well with the high dimensionality of text data.
[21]	Textual	For newly reported bugs, the accuracy of SVM and Neural Net is high. Different measures are used for evaluation.	The accuracy level of different test cases is validated using cross-project validation. Open source projects historical data is used for priority prediction.	By increasing the “K” value performance of KNN decreased. The accuracy of NB is lower than SVM, KNN, and ANN.
[26]	Categorical Data	For analyzing the performance of bug tracking platforms, the proposed approach can be easily integrated.	An accurate model is proposed for the classification of valid bug reports, which can be easily implemented for practical relevance.	It will be a challenge to apply the same methodology to newly born Open Source Software communities.
[27]	Text and Metadata	Decision Tree and Random Forest performance were better than Naive Bayes.	The proposed approach can be used for priority prediction of reported bugs.	For both used datasets Feature set 2(metadata features) performs better than Feature set 1 (textual features)
[1]	Categorical and textual	The analysis shows that in determining blocking bugs important factors are reporter’s experience, a CC list of developers in comments, and comment size.	The proposed model reduces execution time to identify blocking bug, which ultimately helps developers to identify these bugs early.	For some project low recall values achieved.
[6]	Textual	The combination of X-Means clustering and Bayes Net classifier provided the best results.	Performance of classification is improved when applied to clustered data.	Only one feature used for bug prioritization. Multiple features can be used.
[28]	Textual	For the prediction of bug priority level an automated approach is used which is based on emotion words used in problem title.	Results are improved by using the proposed approached.	There are number of other emotion words libraries, which can improve the results.

TABLE 2. Bugs Report example of each dataset.

ID	Class	Component	Summary	OS	Severity	Product
10	P1	Server/DDX	non-XFT builds of Mozilla crash X server when viewing pages with unrecognized unicode characters	All	critical	Mozilla
1006	P2	Project Creation	new project: python-geoclue	All	normal	Free Desktop
1724	P3	UI	small typo in a comment in apply Styles in Abstract CSS Engine	All	trivial	Eclipse
1788	P2	JSP	Merged the jsp parser	All	trivial	NetBeans
3518	P1	Blink>Infra	Bad-cast to blink::WebView from invalid vptr;test_runner	Linux	High	Chromium

is used. A large number of features will increase complexity and obviously running time of algorithms. In the existing model, Naive Bayes provides better results but Naive Bayes does not perform well with high dimension data [4].

F. RESEARCH SCOPE

This research emphasizes the prioritization of bugs. To evaluate the proposed improved model, different bug datasets will be used such as Chromium, Eclipse, NetBeans, Mozilla,

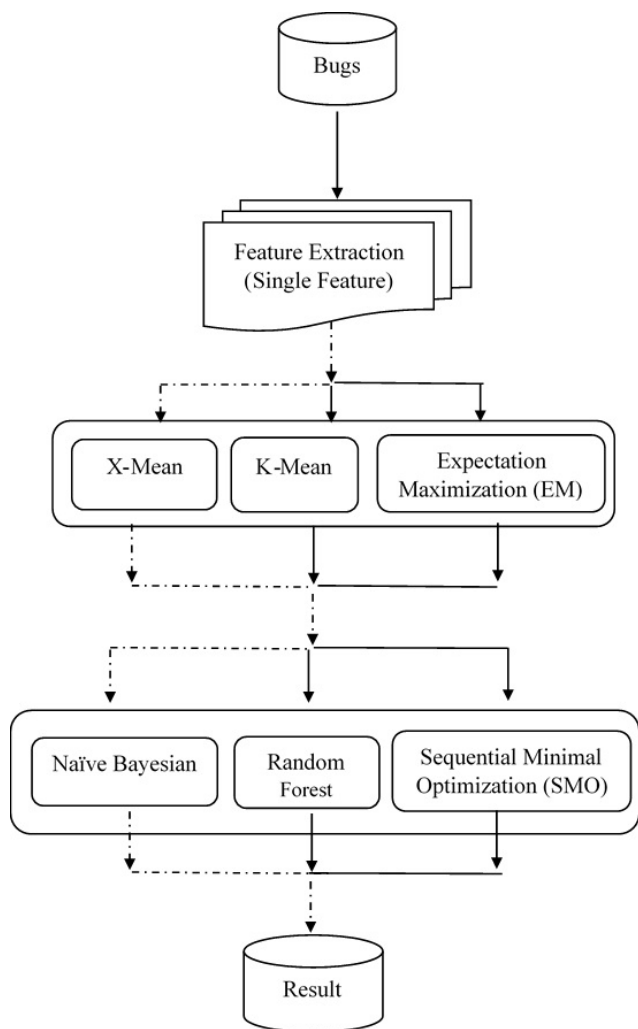


FIGURE 1. Existing research model [6].

and FreeDesktop. Each dataset contains 19917, 10000, 9909, 3398, and 2184 bugs respectively. For a comparison of the proposed model with an existing model of Goyal *et al.* [6], evaluation criteria like accuracy, F-score, precession, and recall will be utilized.

G. EXISTING BUG PRIORITIZATION MODEL

In the existing model single feature is used for bug prioritization [6]. The existing model is shown in Figure 1. Numeric features are extracted from a single feature problem title, which leads to information loss. Three clustering algorithms X-Mean, K-Mean, and Expectation Maximization are used. Classification with clustering performs better results. The clustering result of each algorithm is given to three different classification algorithms Random Forest, Naive Bayes, and Sequential Minimal Optimization. Results of all combinations are compared, and it was found that X-Mean clustering performs better results with Naive Bayes. The dotted line is used in Figure 1 to highlight the best combination.

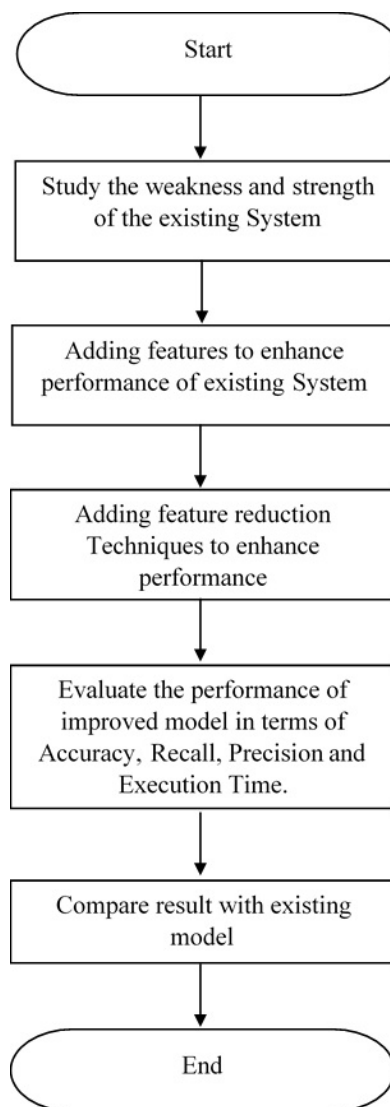


FIGURE 2. Research framework for bug prioritization.

III. METHODOLOGY

This research starts with the study of the existing bug prioritization model; to determine the strengths and weaknesses of the existing model Goyal *et al.* [6]. After the in-depth study of the existing bug prioritization model, few limitations were recognized as explained in the problem statement in Section II-F. The proposed model will overcome the weakness of the existing model. A comparison with the existing model will be carried out in order to determine if the proposed model succeeds to get the best results? In the proposed framework weakness and strengths of the existing bug, the prioritization model is studied. After finding weaknesses new features are added to improve the performance of the existing model. Figure 2 shows the proposed framework.

A. PROPOSED BUG PRIORITIZATION MODEL

The proposed bug prioritization model is shown in Figure 3. Three features are used for bug prioritization, “problem

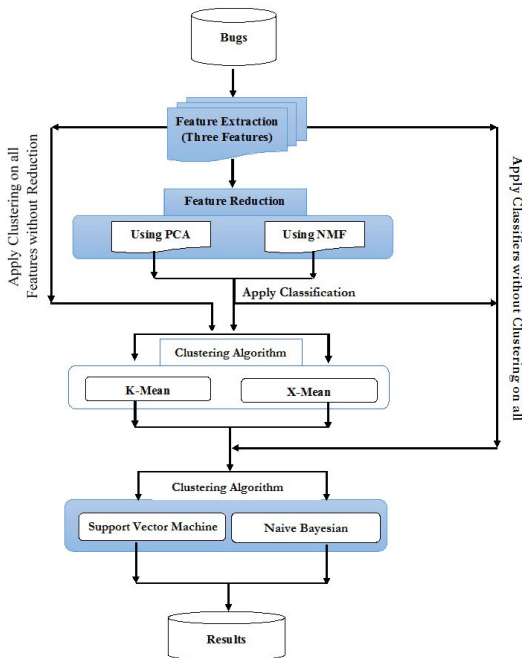


FIGURE 3. Proposed model for bug prioritization.

title”, “component” and “severity”. Modification / New additions including “features reduction techniques” and “classification algorithms” are shaded in 3. In each dataset, 60% data is used for training and 40% data is used for testing. The implementation of the proposed model can be divided into four categories.

- Classification on all features.
 - Clustering on all features and then apply classification on clustered data.
 - Features reduction and then apply classification on reduced features.
 - Feature reduction, then apply clustering on reduced features and finally apply classification on clustered data.
- The proposed model is discussed in subsequent sections.

1) FEATURE EXTRACTION

The bug reports have many features, some are filled during processing the reports, and some are filled by the user at the reporting time. The features we will be used for classification are “problem-title”, “component” and “severity”. Problem Title is a text feature while component and severity are categorical features. The algorithms we will be used for clustering and classification work on numeric features. So, we need to transform these categorical and text features to numeric features, for which below information retrieval techniques are used.

2) PRE-PROCESSING

The summary attribute of the bug report contains the unstructured text. It may have some special characters, other language alphabets, sentences, etc, which make this

inappropriate for any type of analysis. To make this fit for analysis the text should be passed through the following stages.

- **Tokenization:** In this step, text provided by the user in the problem title feature is converted to lower case and then divided into tokens (words). Punctuations, symbols such as hyphens, brackets, and nonalphabetic constructs are removed.
- **Stop Word Removal:** In this step, common words, like is, am, are, the, and, a, with, for, etc are removed. Because these words do not mean anything special.
- **Lemmatization:** Words appearing in the problem title of a bug report can appear in different forms. For example, the word “connect” can appear as “connected”, “connection”, “connections” and “connecting”. With the help of lemmatization, all these words will be converted to their ground word “connect”.
- **Term Frequency-Inverse Data Frequency:** In the end, TF-IDF is applied to the words obtained from the previous step to convert them into numeric features. It is one of the most widely used techniques for processing textual data [30]. It is an information retrieval technique that evaluates a term’s frequency (TF) and its inverse document frequency (IDF). Each word or term has its own TF and IDF score. The product of the TF and IDF scores of a term is called the TF*IDF score of that term. The TF*IDF algorithm is used to assign the importance to a word based on the number of times it appears in the bug report. More importantly, it checks how relevant the word is throughout the bug reports. For a word v in a bug report b , the weight $W_{v,b}$ of word v in a bug report is given by $W_{v,b} = TF_{v,b} \cdot \log(N/DF_v)$ Where $TF_{v,b}$ is the number of occurrences of v in bug report b , DF_v is the number of bug reports containing the word v , and N is the total number of bug reports in a dataset.

3) NON-NEGATIVE MATRIX FACTORIZATION

NMF also is known as positive factorization or non-negative rank factorization has been studied by different researchers in the past few decades [31], [32]. It got popularized when Lee and Seung [33] discovered that with the help of it, images of visual objects can be decomposed into meaningful parts. According to Cichocki and Phan [34] NMF has emerged as a very helpful technique for data mining, information retrieval, blind source separation (BSS), and clustering applications. There are a lot of good algorithms produced by NMF including geometry-based methods [35] and optimization-based methods [34], [36]. NMF belongs to the unsupervised matrix decompositions category in which it is guaranteed that the values of attributes will be not negative [37]. In many applications, Non-negativity is a valid constraint, due to which NMF got success by providing interpretable and meaningful results, even “correct” results sometimes [38]. Matrix decomposition methods such as Singular-Value Decomposition (SVD), Independent Component Analysis (ICA), and Principal Component Analysis (PCA) decomposes the matrix

into positive and negative value matrices, which are very hard to interpret [37]. NMF is different from these decomposition methods, it does not allow negative values in matrix decomposition/factorization. As compared to other dimensionality reduction methods like SVD, We can reconstruct the original matrix from NMF decomposition by using no subtractive and only additive combination of basic elements [39]. NMF decomposes the matrix “M” of size “m * n” into factors “W” and “H” i.e. $M = WH$, where W is of size $m * r$ and H is of size $r * n$. Additionally, the reduced rank r is generally chosen as $(n + m) r < m * n$, hence the compression effect is accomplished. As a result, M is able to be estimated as a linear combination of the vectors of the basis matrix W and gains matrix H. The nonnegative rank may be higher than the usual matrix rank over the real field due to nonnegative constraints.

4) PRINCIPAL COMPONENT ANALYSIS

On a daily basis, k bugs are reported, due to which datasets are increasing and are often difficult to interpret. PCA is one of the most widely used techniques for reducing the dimensionality of such datasets. It increases the interoperability of data with minimum information loss [40]. PCA finds patterns in data, and express the data by highlighting their variations and similarities. It is hard to discover patterns in high-dimensional data sets where graphic representation luxury is not available. PCA is a powerful analysis tool for this data. When patterns are found in data, the number of dimensions can be reduced by using compression. Understanding variance, standard deviation, and covariance are very important for understanding the working of PCA. PCA algorithm is a six-step process.

- 1) Load dataset.
- 2) Find the mean of each data dimension and subtract values from mean.
- 3) Find the covariance matrix of the dataset.
- 4) Find eigenvalues and eigenvector of the covariance matrix.
- 5) Select Components(Eigenvectors) and form a feature vector (matrix of eigenvectors).
- 6) Deriving the new dataset.

5) K-MEANS Clusterink2g

K-means clustering is a kind of unsupervised learning, that is used for unlabeled data. The aim of this algorithm is to classify your objects into the K number of the group based on attributes/features. K is a positive integer number. The grouping is carried out by reducing the sum of distance squares between data in the matching cluster center. Hence, the aim of K-mean clustering is to categorize the data into K groups. Below are the steps of the K-mean algorithm.

- 1) Define the K value. (How many clusters or groups of data are required).
- 2) Randomly select “K” number of centroids from data.
- 3) Calculate the distance of each data item from each centroid.
- 4) Group data items based on minimum centroid distance.

- 5) Repeat steps “c” and “d” until the data item clusters have not been changed.

6) X-MEANS CLUSTERING

X – means is an extended version of the K – mean algorithm. X – mean was first introduced by Yahoo. This algorithm consists of the below steps.

- 1) Traditional K-Mean algorithm
- 2) Improve Structure
- 3) If $K > K_{max}$ stops and reports the best scoring model found during the search. Else go to step 1.

Once the K-mean algorithm executes. In step 2 it is tried to split the resulted centroids into two centroids if possible. There are two approaches used for splitting the final centroids provided by K-mean. Approach one is to pick each centroid one by one, split the selected centroid into two centroids, run K-mean to completion and use Bayesian Information Criterion (BIC) or Akaike Information Criterion (AIC) measure to see if the new resulting model scores better. If it does, accept the new centroid, otherwise return to the previous structure. In the first approach, each centroid is tested for a split which is an expensive process and will need $O(K_{max})$. To overcome this expensive process second approach is to pick half of the centroids according to some heuristic criterion for how promising they are to split. Split them, run K-means, and check if the resulting model scores better than the original using BIC or AIC measure. Accept the split, if it performs better, otherwise return to the previous structure.

7) SUPPORT VECTOR MACHINE

SVM is a supervised machine learning algorithm also known as maximal margin classifier and is used both for regression and classification. The goal of SVM is to discover hyperplane in an N-dimensional area that clearly categorizes the data points. To categorize data points of different classes, multiple hyperplanes exists that can be chosen. Discovering a plane that has maximum margin should be objective, i.e the maximum distance between data points of different categories. Margin and supporting vectors are shown in Figure 4.

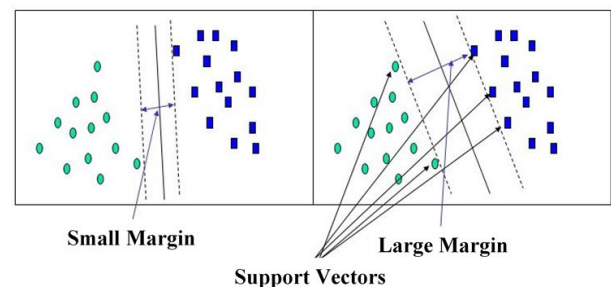


FIGURE 4. Support vectors and margin [41].

8) SUPPORT VECTORS AND HYPERPLANE

The hyperplane is boundaries that help in separating data points of different categories. The number of hyperplanes

is dependent on a number of features. In the case of two features, the hyperplane will be a single line. In the case of three features, a two-dimensional plane will be the hyperplane. Support vectors are data points that are nearer to the hyperplane and affect the position and orientation of the hyperplane. Using these support vectors, the margin of the classifier is maximized. The location of the hyperplane is updated if support vectors are deleted.

9) NAIVE BAYES

Naive Bayes is built on Bayes' theorem with an assumption of independence between indicators. It is a commonly used machine learning classifier due to its effectiveness and simplicity. It assumes the occurrence of a feature in a group/class is not linked to the occurrence of any other feature. Using maximum posterior decision rules in the Bayesian setting, it makes probabilistic classifications. It is commonly used for text classification and spam detection problem. Bayes theorem gives a way of calculating posterior probability $P(c|x)$ from $P(c)$, $P(x)$, and $P(x|c)$ as shown in the following equation:

$$P(c|x) = \frac{P(x|c) \cdot P(c)}{P(x)} \quad (1)$$

$P(c|x)$ is the posterior probability of class (c, target) given predictor (x, attributes). $P(c)$ is the prior probability of a class, $P(x|c)$ is the likelihood which is the probability of predictor given class while $P(x)$ is the prior probability of predictor. The prior probability represents uncertainty before sampling any data. The posterior probability represents uncertainty after sampling data. The probability is the likelihood of an already occurred event that will produce a specific outcome. The events which are going to occur in the future refer to probability, while the events which occurred in the past with known outcomes refer to the likelihood.

B. EXPERIMENTAL STUDY

The following are the research questions of the proposed research work that will be ascertained in the subsequent results section.

- Does the addition of the SVM classifier and increasing the number of features improve the bug prioritization model?
- Does feature reduction techniques improve the performance of the bug prioritization model?

While the following are the research objectives of the proposed research work.

- To propose an improved model for bug prioritization using clustering with classification.
- To evaluate the impact of features reduction techniques on the bug prioritization model.

C. TESTING, COMPARISON, AND ANALYSIS

Recall, Precision, and F-Score are used to test the achievement of the proposed model and are compared with the existing model. The results of the following performance matrices

TABLE 3. Confusion matrix.

Actual Class	Predicted Class	
	Class = Yes	Class = No
Class = Yes	True Positive	False Negative
Class = No	False Positive	True Negative

shall be provided in the results section.

$$Precision = ((RelevantBugs) * 100) / (TotalRetrievedBugs) \quad (2)$$

$$Recall = ((RelevantBugs) * 100) / (TotalBugsinDataset) \quad (3)$$

$$F-Score = 2 * ((Precision * Recall) / (Precision + Recall)) \quad (4)$$

$$Accuracy = ((RelBugs) + (RetBugs)) / (TotalBugs) \quad (5)$$

IV. RESULTS

This section presents results and a comparison of the proposed model with the existing model [6]. The section is organized as: Section IV-A presents evaluation metrics, Section IV-B discusses experimental datasets, Section IV-C presents a comparison and Section IV-D describes chapter summary.

A. EVALUATION METRICS

Evaluation metrics are important parts of a research process for assessment of the results. To evaluate results in this research four types of matrices precision, recall, f-score, and accuracy are used. The performance of our model can be measured using the confusion matrix on a set of test data for which the true values are known. There are four parameters of the confusion matrix, True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). True positives and true negatives are the correctly predicted values while false positives and false negatives are the predicted values having a conflict which actual values. These four parameters are shown in Table 3.

True Positives (TP): These are the correctly predicted positive values which mean that the class of a bug report is "P1" and the predicted class by the model is also "P1".

True Negatives (TN): These are the correctly predicted negative values which mean that the actual class of a bug report is "P2" and the predicted class is also "P2".

False Positives (FP): When the actual class of a bug report is "P3" and the predicted class of a bug report is not "P3".

False Negatives (FN): When the actual class of a bug report is "P2" but the predicted class of a bug report is not "P2".

1) PRECISION

Precision is the ratio of correctly predicted positive bug reports to the total predicted positive bug reports. Precision

TABLE 4. Experimental results for SVM and NB using single feature.

Dataset	SVM				Naive Bayes			
	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	80.00	81.00	80.00	80.76	74.00	69.00	71.00	69.09
Eclipse	94.00	97.00	95.00	96.95	94.00	90.00	92.00	90.00
Free Desktop	81.00	83.00	78.00	82.68	75.00	55.00	60.00	54.93
Mozilla	76.00	83.00	76.00	83.06	71.00	62.00	66.00	62.22
NetBeans	42.00	64.00	51.00	64.48	49.00	30.00	34.00	29.94

shows the usefulness of the system and is calculated through the following equation:

$$Precision = ((RelBugs) * 100) / (TotalRetBugs) \quad (6)$$

2) RECALL

The recall is the ratio of correctly predicted bug reports in a class to all bug reports in a class and is calculated through the following equation:

$$Recall = ((Relevant Bugs) * 100) / (Total Bugs) \quad (7)$$

3) F SCORE

F Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives predictions and is calculated through:

$$F - Score = 2 * (Recall * Precision) / (Recall + Precision) \quad (8)$$

4) ACCURACY

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. The relevant bugs are denoted with Rel Bugs while the retrieved bugs are presented by Ret Bugs. The following expression computes the accuracy:

$$Accuracy = ((Rel Bugs) + (Ret Bugs)) / (Total Bugs) \quad (9)$$

B. EXPERIMENTAL DATA SETS

Datasets used for experimental results are Chromium [42], Eclipse [43], NetBeans [44], Mozilla [45], and Free Desktop [46]. Umer et al. [28], Choudhary [47], and many other researchers used Eclipse in their research. Valdivia Garcia and Shihab [1] used all the five datasets in their research. Zanetti et al. [26] used Eclipse, NetBeans, and Mozilla in their research. Overall Eclipse is the most popular dataset used by many researchers. All the experiments were implemented in Python 3.6.4 Anaconda using the SKlearn library in Microsoft Visual Studio 2015. SQL Server 2014 DBMS (Database Management System) is used for data

storage. All experiments were performed on a personal laptop with Corei7 Quad-Core Processor having 16 GB RAM.

C. RESULTS

This section presents the comparison of the proposed bug prioritization model with the existing bug prioritization model [6]. First results are produced for the existing model, then the proposed model results are produced and, in the end, they are compared.

1) EXISTING MODEL RESULTS USING SINGLE FEATURE

The results of using Naive Bayes and SVM classifiers with a single feature are shown in Table 4. In the second approach, K-Mean clustering is applied to each dataset before classification while in the third approach X-Mean clustering is applied to each dataset before classification. In the existing model Naive Bayes performs better while adding SVM in the existing model, SVM performs better than Naive Bayes.

Similarly, The experimental results of classification with K-Mean and X-Mean clustering are shown in Table 4.3 and 4.4 respectively.

Table 5 presents the experimental results using precision, recall, f-score, and accuracy for K-Mean clustering with SVM and Naive Bayes classification. The columns show the precision, recall, f-score, and accuracy of both approaches i.e. K-Mean with SVM and K-Mean with Naive Bayes. Whereas, the rows of the table show the datasets used. From Table 5 it can be observed that K-Mean with SVM approach outperforms K-Mean with Naive Bayes. It improves the accuracy for all the datasets by a minimum 5.8% to a maximum 22.79%.

Table 6 presents the experimental results using precision, recall, f-score, and accuracy for X-Mean clustering with SVM and Naive Bayes classification. The columns show the precision, recall, f-score, and accuracy of both approaches i.e. X-Mean with SVM and X-Mean with Naive Bayes. Whereas, the rows of the table show the datasets used. It can be observed that X-Mean with SVM approach outperforms X-Mean with Naive Bayes. It improves the accuracy for all the datasets by a minimum 4.8% to a maximum 21.4%.

TABLE 5. Experimental results for K-Mean with SVM & K-Mean with NB using single feature.

Dataset	K-Mean and SVM				K-Mean and Naive Bayes			
	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	91.50	91.75	91.00	91.55	89.25	85.25	87.00	85.29
Eclipse	93.80	96.80	95.20	96.80	94.40	90.80	92.80	91.00
Free Desktop	83.60	85.60	81.60	85.86	79.80	57.20	62.20	56.89
Mozilla	78.40	87.20	81.20	87.06	77.00	71.40	74.00	71.15
NetBeans	52.25	67.75	55.75	67.72	51.75	45.00	47.00	44.93

TABLE 6. Experimental results for X-Mean with SVM & X-Mean with NB using a single feature.

Dataset	X-Mean + SVM				K-Mean and Naive Bayes			
	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	87.78	87.03	85.59	86.96	87.13	79.88	81.31	79.90
Eclipse	96.81	98.34	97.50	98.32	97.19	93.63	95.25	93.53
Free Desktop	75.16	83.16	77.19	83.20	68.53	57.03	56.66	56.98
Mozilla	78.97	88.06	82.78	88.01	77.81	62.50	68.06	62.52
NetBeans	47.41	65.66	53.13	65.74	52.13	44.38	46.63	44.32

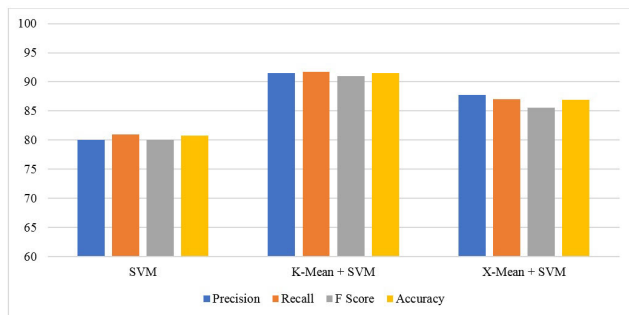


FIGURE 5. SVM comparison for chromium of existing model.

Figure 5 presents the precision, recall, f-score, and accuracy of different algorithmic approaches for chromium. It can be seen from Figure 5 that K-Mean + SVM approach has outperformed X-Mean + SVM and SVM approach.

Figure 6 presents the comparison of different algorithmic approaches for the eclipse. It can be seen from Figure 6 that X-Mean + SVM approach has outperformed K-Mean + SVM and SVM approach.

Figure 7 presents the comparison of different algorithmic approaches for free desktop. It can be seen from Figure 7 that K-Mean + SVM approach has outperformed X-Mean + SVM and SVM approach.

Figure 8 presents the comparison of different algorithmic approaches for Mozilla. It can be seen from 8 that X-Mean + SVM approach has outperformed K-Mean + SVM and SVM approach.

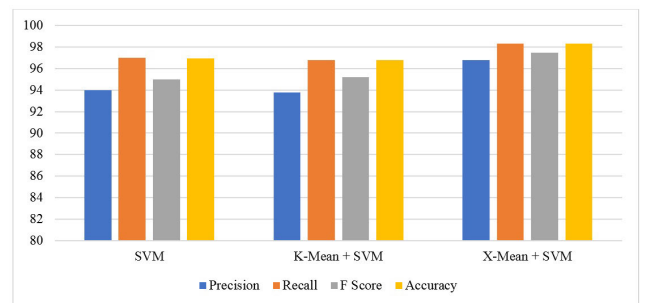


FIGURE 6. SVM comparison for Eclipse of the existing model.

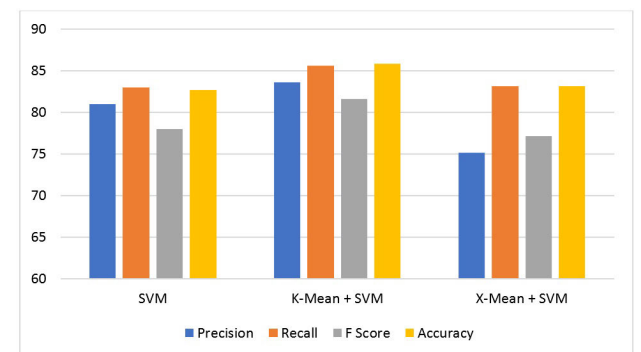


FIGURE 7. SVM comparison for free desktop of the existing model.

Figure 9 presents the comparison of different algorithmic approaches for Net Beans. It can be seen from Figure 9 that K-Mean + SVM approach has outperformed X-Mean + SVM and SVM approach.

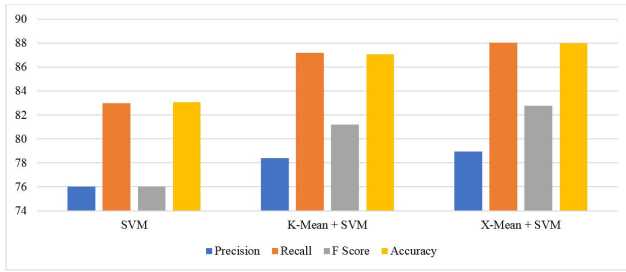


FIGURE 8. SVM comparison for Mozilla of the existing model.

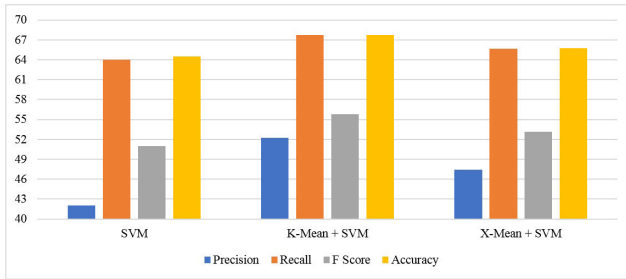


FIGURE 9. SVM comparison for NetBeans of the existing model.

2) RESULTS OF THE PROPOSED MODEL USING THREE FEATURES

In the proposed model three features are used for clustering and classification. The results are divided into two main sections. Each section is further divided into three subsections.

- Classification on all features
 - Direct Classification on all bug reports
 - Classification on K-Mean clustered bug reports
 - Classification on X-Mean clustered bug reports
- Classification on reduced features
 - Direct Classification on all reduced features bug reports
 - Classification on K-Mean clustered bug reports
 - Classification on X-Mean clustered bug reports

Classification on all features: Results of classification on all features for Chromium, Eclipse, Free Desktop, Mozilla and NetBeans with and without clustering using Naive Bayes and SVM Classifier are shown in Tables 7, 8 and 9. Comparison of direct classification, classification with K-Mean and X-Mean of each dataset are shown in Figures 10, 11, 12, 13, and 14 respectively.

Figure 10 presents the comparison of different algorithmic approaches for Chromium. It can be seen from Figure 10 that K-Mean + SVM approach has outperformed X-Mean + SVM and SVM approach.

Figure 11 presents the comparison of different algorithmic approaches for Eclipse. It can be seen from Figure 11 that X-Mean + SVM approach has outperformed K-Mean + SVM and SVM approach.

Figure 12 presents the comparison of different algorithmic approaches for the Free Desktop. It can be seen from

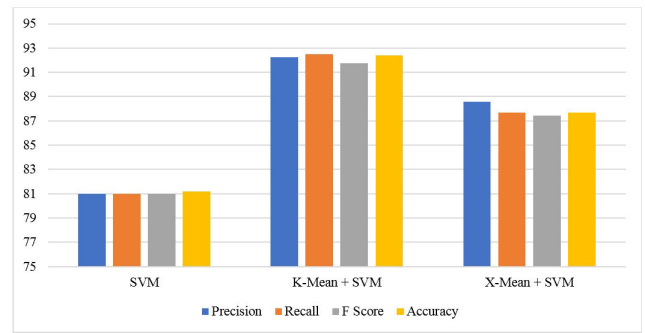


FIGURE 10. SVM comparison for Chromium of the proposed model.

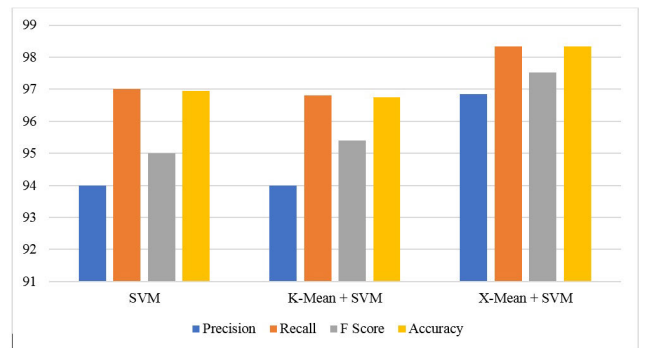


FIGURE 11. SVM comparison for Eclipse of the proposed model.

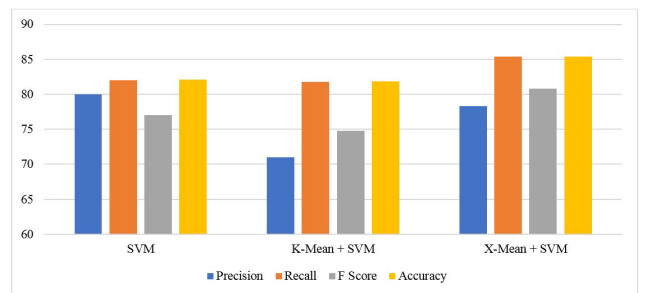


FIGURE 12. SVM comparison for free desktop of the proposed model.

Figure 12 that X-Mean + SVM approach has outperformed K-Mean + SVM and SVM approach.

Figure 13 presents the comparison of different algorithmic approaches for Mozilla. It can be seen from Figure 13 that X-Mean + SVM approach has outperformed K-Mean + SVM and SVM approach.

Figure 14 presents the comparison of different algorithmic approaches for Net Beans. It can be seen from Figure 14 that X-Mean + SVM approach has outperformed K-Mean + SVM and SVM approach.

Table 7 presents the experimental results for SVM and Naive Bayes. Table 8 presents the experimental results of SVM and Naive Bayes with K-Mean clustering while Table 9 presents the experimental results of SVM and Naive Bayes with X-Mean clustering. Overall in all three approaches, SVM outperformed Naive Bayes in terms of precision, recall,

TABLE 7. Experimental results for SVM & Naive Bayes employed on three features.

Dataset	SVM				K-Mean and Naive Bayes			
	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	81.00	81.00	81.00	81.20	74.00	70.00	71.00	70.14
Eclipse	94.00	97.00	95.00	96.95	94.00	91.00	92.00	90.54
Free Desktop	80.00	82.00	77.00	82.11	75.00	56.00	60.00	55.73
Mozilla	69.00	83.00	75.00	82.99	72.00	63.00	67.00	63.25
NetBeans	63.00	65.00	52.00	64.61	49.00	33.00	37.00	32.69

TABLE 8. Experimental results for K-Mean + SVM & K-Mean + NB using three features.

Dataset	K-Mean + SVM				K-Mean and Naive Bayes			
	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	92.25	92.50	91.75	92.40	89.00	85.50	87.00	85.70
Eclipse	94.00	96.80	95.40	96.74	94.00	92.20	93.00	92.20
Free Desktop	71.00	81.80	74.80	81.88	75.20	46.40	53.60	46.31
Mozilla	76.80	85.80	80.20	85.80	76.40	77.60	76.60	77.00
NetBeans	51.25	63.50	51.00	63.40	46.00	40.75	42.75	40.70

TABLE 9. Experimental results for X-Mean + SVM & X-Mean + NB using three features.

Dataset	X-Mean + SVM				K-Mean and Naive Bayes			
	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	88.59	87.69	87.44	87.68	88.72	82.59	83.69	82.56
Eclipse	96.84	98.34	97.53	98.34	97.09	93.78	95.41	93.83
Free Desktop	78.28	85.41	80.78	85.40	75.09	63.34	66.25	63.33
Mozilla	80.47	88.72	84.13	88.67	81.94	78.19	79.53	78.19
NetBeans	48.09	66.31	54.09	66.29	52.31	47.06	49.00	47.00

f-score, and accuracy. In Table 7 minimum 6.41% and maximum 26.38% accuracy is improved. In Table 8 minimum 4.5% and maximum 35.57% accuracy is improved while in Table 9 minimum 4.51% and maximum 22.07% accuracy is improved.

Classification on Reduced Features: PCA and NMF are applied to each dataset for feature reduction. Each dataset features are reduced to 1 and 10 features. Results of classification on reduced features for Chromium, Eclipse, Free Desktop, Mozilla and NetBeans with and without clustering using Naive Bayes and SVM Classifier are shown in the

Tables 10, 11, 12, 13 and 14. Comparison of direct classification on reduced features, classification on K-Mean and X-Mean clustered data reduced with PCA and NMF of each dataset are shown in Figures 10 11 12 13 14 respectively.

Table 10 represents NMF feature reduction followed by SVM and Naive Bayes classifier. Each dataset features are reduced to 1-feature and 10-features using NMF feature reduction and then SVM and NB classifiers are applied (i.e NMF-SVM and NMF-NB). Overall SVM classifier performed better than the Naive Bayes classifier in terms of Precision, Recall, F Score, and Accuracy.

TABLE 10. Experimental results for SVM and Naive Bayes using NMF feature reduction.

NMF		1 - Feature				10 - Features			
Dataset	Classifier	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	SVM	45.00	67.00	54.00	67.29	45.00	67.00	54.00	67.29
Chromium	NB	65.00	35.00	33.00	35.01	74.00	43.00	53.00	42.61
Eclipse	SVM	94.00	97.00	95.00	96.95	94.00	97.00	95.00	96.95
Eclipse	NB	94.00	97.00	95.00	96.95	95.00	32.00	47.00	31.90
Free Desktop	SVM	64.00	80.00	71.00	79.82	75.20	46.40	53.60	46.31
Free Desktop	NB	64.00	80.00	71.00	79.82	76.00	51.00	60.00	51.15
Mozilla	SVM	69.00	83.00	75.00	82.99	69.00	83.00	75.00	82.99
Mozilla	NB	69.00	83.00	75.00	82.99	71.00	27.00	38.00	27.03
NetBeans	SVM	42.00	64.00	51.00	64.48	42.00	64.00	51.00	64.48
NetBeans	NB	42.00	64.00	51.00	64.48	42.00	21.00	23.00	20.76

TABLE 11. Experimental results for SVM and Naive Bayes using PCA feature reduction.

PCA		1 - Feature				10 - Features			
Dataset	Classifier	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	SVM	45.00	67.00	54.00	67.29	45.00	67.00	54.00	67.29
Chromium	NB	51.00	67.00	55.00	66.81	57.00	31.00	35.00	31.09
Eclipse	SVM	94.00	97.00	95.00	96.95	94.00	97.00	95.00	96.95
Eclipse	NB	94.97	97.00	95.00	96.95	95.00	6.00	10.00	6.23
Free Desktop	SVM	64.00	80.00	71.00	79.82	64.00	80.00	71.00	79.82
Free Desktop	NB	67.00	20.00	9.00	19.72	62.00	18.00	11.00	17.89
Mozilla	SVM	69.00	83.00	75.00	82.99	69.00	83.00	75.00	82.99
Mozilla	NB	69.00	83.00	75.00	82.77	68.00	8.00	9.00	7.81
NetBeans	SVM	42.00	64.00	51.00	64.48	42.00	64.00	51.00	64.48
NetBeans	NB	45.00	20.00	9.00	19.81	44.00	8.00	5.00	8.45

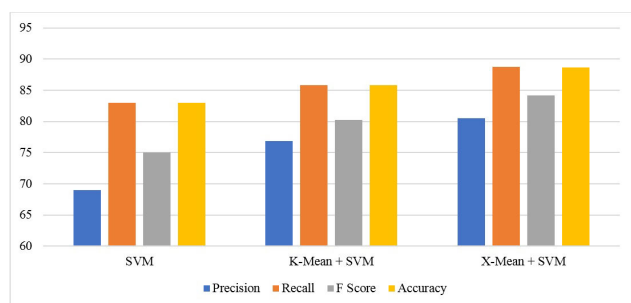


FIGURE 13. SVM comparison for Mozilla of the proposed model.

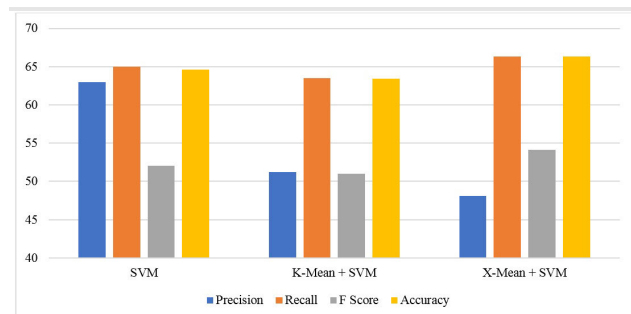


FIGURE 14. SVM comparison for NetBeans of the proposed model.

Table 11 represents PCA feature reduction followed by SVM and Naive Bayes classifier. Each dataset features are reduced to 1-feature and 10-features using PCA feature

reduction and then SVM and NB classifiers are applied (i.e. PCA-SVM and PCA-NB). Overall SVM classifier performed better than the Naive Bayes classifier in terms of Precision, Recall, F Score, and Accuracy. When features are increased

TABLE 12. Experimental results of NMF + K-Mean + SVM & NMF + K-Mean + Naive Bayes.

NMF+KMean		1-Feature				10-Features			
Dataset	Classifier	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	SVM	73.25	78.00	74.75	78.00	75.00	45.75	50.50	45.73
Chromium	NB	69.00	81.50	74.00	81.28	63.00	76.75	68.25	76.68
Eclipse	SVM	94.60	96.80	95.80	97.00	95.60	71.40	78.40	71.24
Eclipse	NB	94.60	97.00	95.80	97.22	95.40	97.40	96.40	97.65
Free Desktop	SVM	66.20	80.80	73.00	80.89	75.20	44.00	52.20	44.03
Free Desktop	NB	66.20	81.20	73.20	81.42	66.20	81.40	73.20	81.38
Mozilla	SVM	74.80	86.00	80.00	86.29	81.60	47.00	55.00	46.91
Mozilla	NB	74.80	86.00	80.00	86.29	78.80	88.40	83.20	88.46
NetBeans	SVM	41.50	64.50	50.50	64.65	61.25	27.50	33.25	27.41
NetBeans	NB	41.50	64.51	50.50	64.65	50.75	70.50	59.00	70.95

TABLE 13. Experimental results of NMF + X-Mean + SVM & NMF + X-Mean + Naive Bayes.

NMF + X-Mean		1 - Feature				10 - Features			
Dataset	Classifier	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	SVM	81.08	86.84	82.83	86.80	79.53	69.64	69.81	69.64
Chromium	NB	72.70	83.70	77.25	83.73	59.64	74.59	65.23	74.56
Eclipse	SVM	96.10	97.26	96.67	97.24	97.63	85.86	89.89	85.88
Eclipse	NB	97.67	98.83	98.23	98.81	97.56	98.76	98.14	98.73
Free Desktop	SVM	67.61	74.25	70.31	74.25	80.56	67.67	70.05	67.65
Free Desktop	NB	73.56	84.92	78.56	84.91	78.81	88.19	82.95	88.15
Mozilla	SVM	82.38	89.39	85.61	89.39	86.36	76.52	79.58	76.48
Mozilla	NB	81.77	89.83	85.45	89.86	87.16	93.00	89.89	92.99
NetBeans	SVM	47.83	66.77	55.30	66.72	56.00	39.64	42.80	39.63
NetBeans	NB	48.19	68.72	56.33	68.65	50.03	69.41	57.63	69.36

from one to 10, NB performance decreases as compared to SVM.

Table 12 represents NMF feature reduction with K-Mean clustering followed by SVM and NB classifier. Each dataset features are reduced to 1-feature and 10-features using NMF. After feature reduction K-Mean clustering is applied on reduced features and at the end SVM and NB classifiers are applied on reduced clustered data. Overall SVM classifier performed better with NMF and K-Mean as compare to Naive Bayes classifier in terms of Precision, Recall, F Score and Accuracy.

Table 13 represents NMF feature reduction with X-Mean clustering followed by SVM and NB classifier. Each dataset features are reduced to 1-feature and 10-features using NMF. After feature reduction, X-Mean clustering is applied to reduced features, and at the end, SVM and NB classifiers

are applied to reduced clustered data. Overall SVM classifier performed better with NMF and X-Mean as compare to the Naive Bayes classifier in terms of Precision, Recall, F Score, and Accuracy.

Table 14 represents PCA feature reduction with K-Mean clustering followed by SVM and NB classifier. Each dataset features are reduced to 1-feature and 10-features using PCA. After feature reduction, K-Mean clustering is applied to reduced features, and at the end, SVM and NB classifiers are applied to reduced clustered data. Overall SVM classifier performed better with PCA and K-Mean as compare to the Naive Bayes classifier in terms of Precision, Recall, F Score, and Accuracy. From the experimental results, it is cleared that Naive Bayes does not perform better in high dimensionality. Naive Bayes performs better on 1-feature as compare to 10-features.

TABLE 14. Experimental results of PCA + K-Mean + SVM & PCA + K-Mean + Naive Bayes.

PCA + K-Mean		1 - Feature				10 - Features			
Dataset	Classifier	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	SVM	73.00	83.00	76.75	83.15	75.25	63.50	61.50	63.39
Chromium	NB	55.75	72.75	62.25	72.92	71.00	81.50	75.00	81.62
Eclipse	SVM	95.20	96.80	96.00	96.91	98.00	68.40	73.20	68.47
Eclipse	NB	95.20	97.60	96.00	97.55	97.60	98.80	98.00	98.78
Free Desktop	SVM	78.40	87.40	82.00	87.31	60.00	23.00	24.00	22.87
Free Desktop	NB	78.40	88.00	82.60	87.86	64.00	80.00	71.00	79.77
Mozilla	SVM	78.60	87.80	82.80	87.85	94.20	82.40	83.60	82.45
Mozilla	NB	78.60	88.40	82.80	88.25	93.80	96.60	95.00	96.59
NetBeans	SVM	44.00	60.50	50.50	60.57	47.00	9.00	6.00	9.41
NetBeans	NB	40.00	63.25	49.00	62.97	42.00	64.00	51.00	64.47

TABLE 15. Experimental results of PCA + X-Mean + SVM & PCA + X-Mean + Naive Bayes.

PCA + X-Mean		1 - Feature				10 - Features			
Dataset	Classifier	Precision	Recall	F Score	Accuracy	Precision	Recall	F Score	Accuracy
Chromium	SVM	76.71	85.16	79.88	85.15	84.67	84.53	83.66	84.55
Chromium	NB	72.33	78.17	74.11	78.20	85.77	85.05	84.39	85.07
Eclipse	SVM	97.29	98.68	97.96	98.65	98.14	87.84	90.45	87.90
Eclipse	NB	97.29	98.68	97.96	98.65	97.64	98.83	98.22	98.78
Free Desktop	SVM	73.36	77.78	73.19	77.73	76.47	78.50	76.25	78.49
Free Desktop	NB	68.75	81.38	74.25	81.28	79.95	87.38	83.23	87.37
Mozilla	SVM	80.11	81.92	79.66	81.88	89.73	87.66	88.31	87.67
Mozilla	NB	78.42	87.22	82.53	87.68	85.81	86.83	85.75	86.80
NetBeans	SVM	57.48	70.61	62.02	70.57	69.91	70.56	68.13	70.49
NetBeans	NB	56.23	72.48	62.77	72.46	63.75	68.83	64.09	68.85

In Table 15 represents PCA feature reduction with X-Mean clustering followed by SVM and NB classifier. Each dataset features are reduced to 1-feature and 10-features using PCA. After feature reduction, X-Mean clustering is applied to reduced features, and at the end, SVM and NB classifiers are applied to reduced clustered data. Overall SVM classifier performed better with PCA and X-Mean as compare to the Naive Bayes classifier in terms of Precision, Recall, F Score, and Accuracy.

D. IMPROVEMENT COMPARISON

Comparison is divided into the following categories.

- Existing Model vs Proposed Model all features
- Existing Model vs Proposed Model reduced features

- Proposed Model all features vs Proposed Model reduced features.

1) EXISTING MODEL VS PROPOSED MODEL (ALL FEATURES)
 Experimental results are divided into three categories. Improvement in Direct classification, improvement in Classification with K-Mean clustering, and improvement in classification with X-Mean clustering. Overall SVM classifier achieved the highest accuracy with X-Mean and K-Mean clustering. Dataset wise improvement is shown in Tables 16, 17 and 18.

Table 16 represents proposed model direct classification improvement from the existing model. Maximum 34.67% accuracy is improved by SVM.

TABLE 16. Improvement in direct classification.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	7.00	12.00	10.00	12.11
Eclipse	-	7.00	3.00	6.95
Free Desktop	5.00	27.00	17.00	27.18
Mozilla	2.00	21.00	9.00	20.77
NetBeans	14.00	35.00	18.00	34.67

TABLE 17. Improvement of classification with K-Mean.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	3.00	7.25	4.75	7.11
Eclipse	0.40	6.00	2.60	5.74
Free Desktop	8.80	24.60	12.60	24.99
Mozilla	0.20	14.40	6.20	14.65
NetBeans	0.50	18.50	4.00	18.47

TABLE 18. Improvement of classification with X-Mean.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	1.46	7.81	6.13	7.78
Eclipse	0.35	4.71	2.28	4.81
Free Desktop	9.75	28.38	24.12	28.42
Mozilla	2.66	26.22	16.07	26.15
NetBeans	4.04	21.93	7.46	21.97

TABLE 19. Improvement in direct classification.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	29.00	2.00	17.00	1.80
Eclipse	--	7.00	3.00	6.95
Free Desktop	11.00	25.00	11.00	24.89
Mozilla	2.00	21.00	9.00	20.77
NetBeans	7.00	34.00	17.00	34.54

Table 17 represents the proposed model with K-Mean clustering improvement from the existing model. Maximum 24.99% accuracy is improved by K-Mean + SVM.

Table 18 represents the proposed model with X-Mean clustering improvement from the existing model. Maximum 28.42% accuracy is improved by X-Mean + SVM.

2) EXISTING MODEL VS PROPOSED MODEL (REDUCED FEATURES)

Experimental results are divided into three categories. Improvement in Direct classification, improvement in Classification with K-Mean clustering and improvement in classification with X-Mean clustering. Table 4.17 represents proposed model improvement from the existing model using PCA + SVM. Maximum 34.54% accuracy is improved with PCA + SVM.

Table 20 represents proposed model using NMF (1-Feature) + K-Mean clustering followed by SVM classifier, improvement from the existing mode. Maximum 34.71% accuracy is improved.

Table 21 represents the proposed model using NMF (1-Feature) + X-Mean clustering followed by the SVM classifier, an improvement from the existing model. A maximum 38.71% accuracy is improved.

Table 22 represents proposed model using NMF (10-Feature) + K-Mean clustering followed by SVM classifier. Maximum 41.01% accuracy is improved using NMF (10-Feature) + K-Mean + SVM.

TABLE 20. Improvement in classification with NMF (1-Feature) + K-Mean + SVM.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	5.00	12.50	3.00	12.19
Eclipse	0.60	7.00	3.80	7.22
Free Desktop	8.80	26.20	13.20	26.49
Mozilla	3.80	24.00	14.00	24.07
NetBeans	7.50	34.51	16.50	34.71

TABLE 21. Improvement in classification with NMF (1-Feature) + X-Mean + SVM.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	1.30	14.70	6.25	14.64
Eclipse	3.67	8.83	6.23	8.81
Free Desktop	1.44	29.92	18.56	29.98
Mozilla	10.77	27.83	19.45	27.64
NetBeans	0.81	38.72	22.33	38.71

TABLE 22. Improvement in classification with NMF (10-Feature) + K-Mean + SVM.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	11.00	7.75	2.75	7.59
Eclipse	1.40	7.40	4.40	7.65
Free Desktop	8.80	26.40	13.20	26.45
Mozilla	7.80	26.40	17.20	26.24
NetBeans	1.75	40.50	25.00	41.01

TABLE 23. Improvement in classification with NMF (10-Feature) + X-Mean + SVM.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	14.36	5.59	5.77	5.47
Eclipse	3.56	8.76	6.14	8.73
Free Desktop	3.81	33.19	22.95	33.22
Mozilla	16.16	31.00	23.89	30.77
NetBeans	1.03	39.41	23.63	39.42

TABLE 24. Improvement in classification with PCA (1-Feature) + K-Mean + SVM.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	18.25	3.75	8.75	3.83
Eclipse	1.20	7.60	4.00	7.55
Free Desktop	3.40	33.00	22.60	32.93
Mozilla	7.60	26.40	16.80	26.03
NetBeans	(9.00)	33.25	15.00	33.03

Table 23 represents proposed model using NMF (10-Feature) + X-Mean clustering followed by SVM classifier. Maximum 39.42% accuracy is improved using NMF (10-Feature) + K-Mean + SVM classifier.

Table 24 represents proposed model using PCA (1-Feature) + K-Mean clustering followed by SVM classifier. Maximum 33.03% accuracy is improved using PCA (1-Feature) followed by K-Mean and SVM classifier.

Table 25 represents proposed model using PCA (1-Feature) + X-Mean clustering followed by SVM classifier. Maximum 42.52% accuracy is improved using PCA (1-Feature) followed by X-Mean and SVM classifier.

Table 26 represents proposed model using PCA (10-Feature) + K-Mean clustering followed by SVM classifier. Maximum 34.53% accuracy is improved using PCA (10-Feature) followed by K-Mean and SVM classifier.

TABLE 25. Improvement in classification with PCA (1-Feature) + X-Mean + SVM.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	1.67	9.17	3.11	9.11
Eclipse	3.29	8.68	5.96	8.65
Free Desktop	6.25	26.38	14.25	26.35
Mozilla	7.42	25.22	16.53	25.46
NetBeans	7.23	42.48	28.77	42.52

TABLE 26. Improvement in classification with PCA (10-Feature) + K-Mean + SVM.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	3.00	12.50	4.00	12.53
Eclipse	3.60	8.80	6.00	8.78
Free Desktop	11.00	25.00	11.00	24.84
Mozilla	22.80	34.60	29.00	34.37
NetBeans	7.00	34.00	17.00	34.53

TABLE 27. Improvement in classification with PCA (10-Feature) + X-Mean + SVM.

Dataset	Precision	Recall	FScore	Accuracy
Chromium	11.77	16.05	13.39	15.98
Eclipse	3.64	8.83	6.22	8.78
Free Desktop	4.95	32.38	23.23	32.44
Mozilla	14.81	24.83	19.75	24.58
NetBeans	14.75	38.83	30.09	38.91

Table 27 represents proposed model using PCA (10-Feature) + X-Mean clustering followed by SVM classifier. Maximum 38.91% accuracy is improved using PCA (10-Feature) followed by X-Mean and SVM classifier. Overall SVM classifier performed better or equivalent than Naïve Bayes in every combination of feature reduction and clustering algorithms.

3) PROPOSED MODEL (ALL FEATURES) VS PROPOSED MODEL (REDUCED FEATURES)

Overall “reduced features” proposed model performed better than “all features” proposed model. In both, approaches SVM performed outclass than Naïve Bayes. With “all features” SVM performed better with K-Mean and X-Mean clustering. On average K-Mean + SVM performed better. With “reduced features” SVM performed better with NMF (1-feature) + X-Mean, NMF (10-feature) + X-Mean, PCA (1-feature) + X-Mean and PCA(10-feature) + K-Mean. From the experimental results, it is clear that the reduced feature proposed model performed better than all feature proposed models. Which reduced features execution time algorithms are minimum.

V. CONCLUSION

This research focuses on predicting the priority of reported bugs with the help of data mining algorithms. For this research, we proposed certain changes in the existing model by adding two new features, classification algorithm, and feature reduction techniques.

This research is mainly focused on two aims, the first one is to determine the limitations of the existing bug prioritization model, and the second aim is to solve these limitations by introducing a new improved bug prioritization model.

In order to accomplish these two aims, we have defined two objectives of this research, which are given below:

Objective 1: To propose an improved model for bug prioritization using clustering with classification. For the purpose of achieving this objective X-Mean and K-Mean algorithm are used for clustering and then the SVM algorithm is applied for classification. Experiments show that the proposed model produced a much better result than the existing model in terms of precision, recall, f score, and accuracy.

Objective 2: To evaluate the impact of features reduction techniques on the bug prioritization model. For feature reduction, two algorithms are used PCA and NMF. Experiments show that the feature reduction techniques improved results in terms of precision, recall, f score, and accuracy. Eclipse and FreeDesktop performed better with NMF reduction while Mozilla and NetBeans performed better with PCA reduction which ultimately improved the running time of classification and clustering algorithms. Summary of research contributions The SVM classifier is proposed for predicting the priority of a bug report. SVM classifier performs outclass in high dimensional data. Two new feature “component” and “severity” are added for the classification of the bug report. Also, feature reduction techniques are added for dimensional reduction to improve the running time of algorithms. This research is focused on improving the quality of the bug prioritization model. In this regard, the research contributions are summarized as follows:

- Three features (component, severity, and summary) are used for predicting the priority of bug reports.
- Feature Reduction Techniques are added to improve running time.
- We have replaced Naive Bayes with the SVM classifier to predict the priority of bug reports.

Future directions: Since our proposed model is a combination of feature reduction techniques, clustering, and classification algorithms and it produced significantly better results. We have used five different datasets for our research. The same model can be applied to other datasets like Open Office, Maemo, and GNOME, etc. The same approach of clustering with classification can be used in other research areas as well to improve their results. Besides, research can be conducted in the following directions as well:

- In our proposed work three features (problem title, component, and severity) are used, new features can also be incorporated to enhance the results. There are a lot of other features related to a bug report.
- Neural networks can be incorporated into the proposed model to further improve the results.
- We have used feature reduction techniques which are time-consuming. In the future instead of feature reduction, the feature selection approach can be applied.
- We have applied feature reduction followed by clustering algorithms, which is further followed by classification algorithms for bug prioritization. The same approach can also be applied for Predicting bug severity, identifying valid, invalid, and duplicate bug reports,

assigning experienced developers to newly reported bug reports, and finding blocking bug reports.

- The same approach of clustering with classification can be used in security systems.
- This approach may also be helpful in the bioinformatics area.
- The same approach can also be applied to the image processing field.

REFERENCES

- [1] H. Valdivia Garcia and E. Shihab, "Characterizing and predicting blocking bugs in open source projects," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 72–81.
- [2] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–35, Aug. 2011.
- [3] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *Proc. 15th Eur. Conf. Softw. Maintenance Reeng.*, Mar. 2011, pp. 249–258.
- [4] J. Kanwal and O. Maqbool, "Bug prioritization to facilitate bug report triage," *J. Comput. Sci. Technol.*, vol. 27, no. 2, pp. 397–412, Mar. 2012.
- [5] A. Nigam, B. Nigam, C. Bhaisare, and N. Arya, "Classifying the bugs using multi-class semi supervised support vector machine," in *Proc. Int. Conf. Pattern Recognit., Informat. Med. Eng. (PRIME)*, Mar. 2012, pp. 393–397.
- [6] N. Goyal, N. Aggarwal, and M. Dutta, "A novel way of assigning software bug priority using supervised classification on clustered bugs data," in *Advances in Intelligent Informatics*. New York, NY, USA: Springer, 2015, pp. 493–501.
- [7] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, and H. Shah, "A survey on bug prioritization," *Artif. Intell. Rev.*, vol. 47, no. 2, pp. 145–180, Feb. 2017.
- [8] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: An empirical study of Microsoft windows," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, vol. 1, 2010, pp. 495–504.
- [9] R. Vilalta, M.-K. Achari, and C. F. Eick, "Class decomposition via clustering: A new framework for low-variance classifiers," in *Proc. 3rd IEEE Int. Conf. Data Mining*, 2003, pp. 673–676.
- [10] L. Candillier, I. Tellier, F. Torre, and O. Bousquet, "Cascade evaluation of clustering algorithms," in *Proc. Eur. Conf. Mach. Learn.* New York, NY, USA: Springer, 2006, pp. 574–581.
- [11] S. Kim and M. D. Ernst, "Prioritizing warning categories by analyzing software history," in *Proc. 4th Int. Workshop Mining Softw. Repositories (MSR:ICSE Workshops)*, May 2007, p. 27.
- [12] S. Kim and M. D. Ernst, "Which warnings should i fix first?" in *Proc. 6th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, 2007, pp. 45–54.
- [13] T. Kremenek and D. Engler, "Z-ranking: Using statistical analysis to counter the impact of static analysis approximations," in *Proc. Int. Static Anal. Symp.* New York, NY, USA: Springer, 2003, pp. 295–315.
- [14] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. 28th Int. Conf. Softw. Eng.*, 2006, pp. 361–370.
- [15] J. Anvik and G. C. Murphy, "Determining implementation expertise from bug reports," in *Proc. 4th Int. Workshop Mining Softw. Repositories (MSR:ICSE Workshops)*, 2007, p. 2.
- [16] G. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in *Proc. 16th Int. Conf. Softw. Eng. Knowl. Eng.*, 2004, pp. 1–6.
- [17] S. N. Ahsan, J. Ferzund, and F. Wotawa, "Automated software bug triage system (BTS) based on latent semantic indexing and support vector machine," in *Proc. 4th Int. Conf. Softw. Eng. Adv.*, Sep. 2009, pp. 216–221.
- [18] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Proc. 7th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, May 2010, pp. 1–10.
- [19] K. K. Chaturvedi and V. B. Singh, "Determining bug severity using machine learning techniques," in *Proc. CSI 6th Int. Conf. Softw. Eng. (CONSEG)*, Sep. 2012, pp. 1–6.
- [20] Y. Yao, Y. Liu, Y. Yu, H. Xu, W. Lv, Z. Li, and X. Chen, "K-SVM: An effective SVM algorithm based on K-means clustering," *J. Comput.*, vol. 8, no. 10, pp. 2632–2639, Oct. 2013.
- [21] M. Sharma, P. Bedi, K. K. Chaturvedi, and V. B. Singh, "Predicting the priority of a reported bug using machine learning techniques and cross project validation," in *Proc. 12th Int. Conf. Intell. Syst. Design Appl. (ISDA)*, Nov. 2012, pp. 539–545.
- [22] M. Y. Javed and H. Mohsin, "An automated approach for software bug classification," in *Proc. 6th Int. Conf. Complex, Intell., Softw. Intensive Syst.*, Jul. 2012, pp. 414–419.
- [23] A. Kaur and S. G. Jindal, "Text analytics based severity prediction of software bugs for apache projects," *Int. J. Syst. Assurance Eng. Manage.*, vol. 10, no. 4, pp. 765–782, Aug. 2019.
- [24] A. Kukkar, R. Mohana, A. Nayyar, J. Kim, B.-G. Kang, and N. Chilamkurti, "A novel Deep-Learning-Based bug severity classification technique using convolutional neural networks and random forest with boosting," *Sensors*, vol. 19, no. 13, p. 2964, Jul. 2019.
- [25] L. A. F. Gomes, R. D. S. Torres, and M. L. Côrtes, "Bug report severity level prediction in open source software: A survey and research opportunities," *Inf. Softw. Technol.*, vol. 115, pp. 58–78, Nov. 2019.
- [26] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks: A case study on four open source software communities," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 1032–1041.
- [27] M. Alenezi, K. Magel, and S. Banitaan, "Efficient bug triaging using text mining," *J. Softw.*, vol. 8, no. 9, pp. 2185–2190, Sep. 2013.
- [28] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," *IEEE Access*, vol. 6, pp. 35743–35752, 2018.
- [29] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *Proc. 22nd IEEE/ACM Int. Conf. Automated Softw. Eng.*, Nov. 2007, pp. 34–43.
- [30] U. Erra, S. Senatore, F. Minnella, and G. Caggianese, "Approximate tf-idf based on topic extraction from massive message stream using the gpu," *Inf. Sci.*, vol. 292, pp. 143–161, Jan. 2015.
- [31] J.-C. Chen, "The nonnegative rank factorizations of nonnegative matrices," *Linear Algebra Appl.*, vol. 62, pp. 207–217, Nov. 1984.
- [32] S. L. Campbell and G. D. Poole, "Computing nonnegative rank factorizations," *Linear Algebra Appl.*, vol. 35, pp. 175–182, Feb. 1981.
- [33] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, Oct. 1999.
- [34] A. Cichocki and A.-H. Phan, "Fast local algorithms for large scale non-negative matrix and tensor factorizations," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. 92, no. 3, pp. 708–721, 2009.
- [35] J. Li and J. M. Bioucas-Dias, "Minimum volume simplex analysis: A fast algorithm to unmix hyperspectral data," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, vol. 3, Jul. 2008, pp. III-250–III-253.
- [36] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2001, pp. 556–562.
- [37] A. K. Ch, S. M. Dias, and N. J. Vieira, "Knowledge reduction in formal contexts using non-negative matrix factorization," *Math. Comput. Simul.*, vol. 109, pp. 46–63, Mar. 2015.
- [38] K. Huang, N. D. Sidiropoulos, and A. Swami, "Non-negative matrix factorization revisited: Uniqueness and algorithm for symmetric decomposition," *IEEE Trans. Signal Process.*, vol. 62, no. 1, pp. 211–224, Jan. 2014.
- [39] R. Divya, C. A. Kumar, S. Saijanani, and M. Priyadarshini, "Deceiving communication links on an organization email corpus," *Malaysian J. Comput. Sci.*, vol. 24, no. 1, pp. 17–33, 2017.
- [40] C. O. S. Sorzano, J. Vargas, and A. P. Montano, "A survey of dimensionality reduction techniques," 2014, *arXiv:1403.2877*. [Online]. Available: <http://arxiv.org/abs/1403.2877>
- [41] *Support Vector Machine, Introduction to Machine Learning Algorithms*. Accessed: Sep. 30, 2018. [Online]. Available: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
- [42] Chromium. *Chromium Bugs Dataset*. (May 2020). [Online]. Available: <https://bugs.chromium.org/p/chromium/issues/list>
- [43] eclipse. (May 2020). *Eclipse Bugs Dataset*. [Online]. Available: <https://bugs.eclipse.org/bugs/>
- [44] Netbeans. (May 2020). *Netbeans Bugs Dataset*. [Online]. Available: <https://netbeans.org/bugzilla/query.cgi?format=advanced>
- [45] Mozilla. (May 2020). *Mozilla Bugs Dataset*. [Online]. Available: https://github.com/ansymo/msr2013-bug_dataset/tree/master/data
- [46] Free Desktop. (May 2020). *Free Desktop Bugs Dataset*. [Online]. Available: <https://bugs.freedesktop.org>
- [47] P. A. Choudhary, "Neural network based bug priority prediction model using text classification techniques," *Int. J. Adv. Res. Comput. Sci.*, vol. 8, no. 5, pp. 1315–1319, 2017.

...