Soft Computing and Machine Intelligence

*Soft Computing and Machine Intelligence Journal, Vol (1), Issue (1), 2021*

# Towards Effective Analysis and Tracking of Mozilla and Eclipse Defects using Machine Learning Models based on Bugs Data

**Zohaib Hassan [a]*, Naeem Iqbal [a] and Abnash Zaman [b],**

[a] FSRA&IT Solutions Providing Organization Peshawar, Pakistan
[b] Faculty of Bioinformatics Shaheed Benazeer Bhutto Women University Peshawar, Pakistan
* Corresponding author: Dev.zohaibmrt@hotmail.com

**Abstract:** Analysis and Tracking of bug reports is a challenging field in software repositories mining. It is one of the fundamental ways to explores a large amount of data acquired from defect tracking systems to discover patterns and valuable knowledge about the process of bug triaging. Furthermore, bug data is publically accessible and available of the following systems, such as Bugzilla and JIRA. Moreover, with robust machine learning (ML) techniques, it is quite possible to process and analyze a massive amount of data for extracting underlying patterns, knowledge, and insights. Therefore, it is an interesting area to propose innovative and robust solutions to analyze and track bug reports originating from different open source projects, including Mozilla and Eclipse. This research study presents an ML-based classification model to analyze and track bug defects for enhancing software engineering management (SEM) processes. In this work, Artificial Neural Network (ANN) and Naive Bayesian (NB) classifiers are implemented using open-source bug datasets, such as Mozilla and Eclipse. Furthermore, different evaluation measures are employed to analyze and evaluate the experimental results. Moreover, a comparative analysis is given to compare the experimental results of ANN with NB. The experimental results indicate that the ANN achieved high accuracy compared to the NB. The proposed research study will enhance SEM processes and contribute to the body of knowledge of the data mining field.

**Keywords:** Bugs Tracking; Mozilla; Eclipse; Machine Learning; Artificial Neural Network; Naive Bayesian

## 1. Introduction

Software development projects involve the use of a wide range of tools to produce a software artifact, and as a result, the history of any given software development may be distributed across a number of such tools. Recent research in this area [1] has described the different types of artifacts that can be used to reconstruct a software project's history. These include, but may not be limited to, the source code itself, source code management systems, issue tracking systems, messages between developers and users, meta-data about the projects, including the artifacts produced from each phase of the Software Development life cycle, and usage data. As shown below in Figure 1.

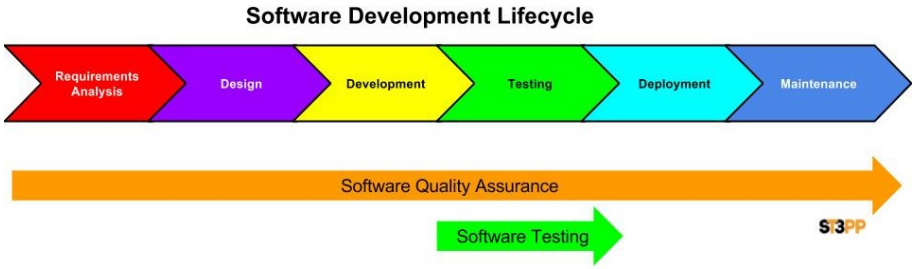


**Figure 1. Software Development Process Life Cycle.**

The history of these artifacts, pertaining to different categories of the Software, is available in huge datasets in the shape of open source systems (OSS). For instance, Software programs, executions potentially (e.g., when program execution traces are turned on) generate huge amounts of data. However, such data sets are rather different from the datasets generated from nature or collected from video cameras since they represent human creativity in different phases such as designs and codes. It is important to mine such data to monitor program execution status, improve system performance, isolate software bugs, detect software plagiarism, analyze programming system faults, and recognize system malfunctions. Data mining for software engineering can be partitioned into static analysis and dynamic/stream analysis, based on whether the system can collect traces beforehand for post-analysis or it must react in real-time to handle online data. Different methods have been developed in this domain by integrating and extending the methods developed in machine learning, data mining, pattern recognition, and statistics. For example, statistical analysis (such as hypothesis testing) approaches [2], can be performed on program execution traces to isolate the bugs' locations that distinguish program success runs from the failed ones. The application of data mining techniques to the software engineering domain and proposing a novel method/algorithm(s) is a research area that can further study/investigate.

## 2. Related work

As background for this work, we discuss various approaches. Common problems that software development organizations encounter in instituting measurement programs include too much data collected, not the right data collected, and preliminary analysis of the collected data. This leads to numerous problems, including the decreased cost-effectiveness of the measurement program and disillusionment about metrics on the part of developers and managers. The end result is often the eventual failure of the measurement program as a whole. In response to such problems, several structured approaches to software measurement have been developed and are used in organizations. These approaches are referred to as "goal-oriented" approaches because they use goals, objectives, strategies, or other mechanisms to systematically guide the choice of data to be collected and analyzed. In [3], they have used CVSgrab to analyze the ArgoUML and PostgreSQL repositories. By clustering the related resources, they generated the projects' evolution based on the clustered file types. Useful conclusions can be drawn by careful manual analysis of the generated visualized project development histories. For example, they discovered that there was only one author for each significant initial contribution in both projects.

**Table 1. Mining Approaches Used in Software Maintenance.**

| Mining Approach | Input Data | Data Analysis Results |
|---|---|---|
| Searching/ matching [4,5] | SCM (bug reports, bug fixes) | Identification of bug introducing changes |
| Clustering [6], | Source code | Extract significant patterns from the system source code groups of similar classes, methods, data |
| Clustering [3] , | SCM, source code | Patterns in history and the development process |
| Clustering [7], | Source code | System modules |
| Clustering [8], Frequent pattern mining and Association Rules | Source code | Structure clone and their categorization |
| Classification [9], | Commits (SCRs) | classes of commits |
| Classification [10], | Source code | FP & NFP modules Classifier |
| Classification [11], | CVS and Bugzilla | Stability of prediction models |
| Frequent pattern mining [12], and Association Rules | Source code | Prediction of failures, correlations between entities identification of additions, modifications, deletions of syntactic entities |
| Frequent pattern mining and association rules [13], | Software libraries | Reused patterns |
| Frequent pattern mining and Association rules [14], | Source code of legacy system | Design alternatives |
| Frequent pattern mining and association rules [15], | Instantiation code of the software | Usage changes |
| Regression, Classification [16], | SCM issue tracking database | Functionality analysis |
| Classification based on Statistics Differencing[3], | Source code | Syntactic and semantic changes |
| Mining based on Statistics, CVS.' annotations [4], | the version history of source code, classes | Syntax & semantic – hidden dependencies |
| Hidden dependencies CVS annotations [5], | Bug & comments modification request | Syntax & semantic file coupling |
| Mining via Heuristic [6], | CVS annotation heuristics | Candidate entities for change |

## 3. Proposed methodology

Research is a highly complex and subtle human activity, which may be difficult, if not impossible, to formulate formally. Nevertheless, some lessons and general principles can be learnt from the experience of scientists. There are some basic principles and techniques that are commonly used in most types of scientific investigations. It is this commonality that can make it possible to extend the scientific research methods to our research. I will adopt the model of the research process of Garziano et al. [7], and combine it with other models [8]. It is possible to combine several phases into one or divide one phase into more detailed steps. The division between phases is not clear-cut. The research methodology will not follow a rigid sequencing of the phases. Iteration of different phases may be necessary [7]. Two effective data mining techniques can be applied to the data sets to deduce the research's desired results. These are: 1) Supervised learning is mainly performed through Classification and Prediction. 2) Unsupervised learning is mainly performed through Clustering. In the specific approach, clustering techniques are used to analyze the input data and provide a rough grasp of the maintenance engineer's software system. Clustering produces overviews of systems by creating mutually exclusive groups of classes, member data, methods based on their similarities. Furthermore, they concluded that PostgreSQL did not start from scratch but was built at some previous project. An interesting evolution of this work could be a more automated way of concluding the development

history, like extracting clusters labels, mapping them to the taxonomy of development processes, and automatically extracting the development phases with comments emerging from taxonomy concepts. We will follow the following steps as shown in the block diagram 2:

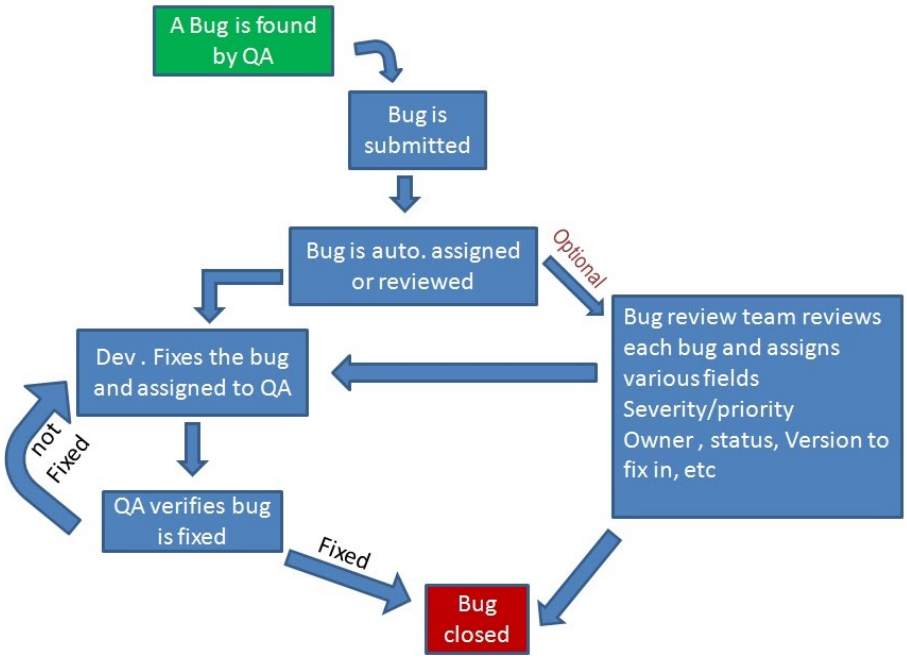


**Figure 2. Block Diagram of the Proposed System.**

The proposed methodology carried out the following steps: Data Analysis Collection of dataset Pre-processing dataset Data Visualization Features Engineering Features Normalization Min-Max Normalization Training/Testing Classifiers ANN NB

## 4. Experimental results and discussions

This section presents the results obtained from the experimentation environment. In this work, the following machine learning (ML) classifiers, i.e., ANN, and NB are trained and tested on the prepared dataset. The following Table 2 is used to present the implementation and experimental setup of the proposed work.

**Table 2. Implementation and experimentation setup of the proposed work.**

| System Component | Description |
|---|---|
| Operating System | Microsoft Windows 8.1 (64-bits) |
| CPU | Intel ®Core ™ i3-2130 CPU at 3.40 GHz |
| Primary Memory | 8 GB RAM |
| Programming Language | Python |
| IDE (Platform) | PyCharm |
| Data Storage | MS Excel, MySQL |
| Classification Tool | WEKA |

In this paper, the following datasets are considered, such as Eclipse and Mozilla Defect Tracking (MDK), which contains bug reports of five popular products. In the Eclipse dataset, we considered the following five products, such as JDT, PDE, Eclipse platform, and CDT. Moreover, the dataset covers each bug report's attributes and includes all the updated lists of each considered bug report. The following ML classifiers are implemented to detect the bug and classify it into the desired category. In this work, we used the k-fold cross-validation (where the value of k is 10) to validate the results

obtained from implemented classifiers [17]. The performances of the proposed classifiers are evaluated and tested based on the following ML measures, such as accuracy (AC), recall (RE), Precision (PR), and F-measure (FM) [18]. All these performance measures are utilized to find the best classifier among implemented classifiers. In this work, we utilized the default parameters for the classification process in WEKA [19]. Figure 3 shown the classification results obtained using ANN and NB classifiers. The classification results of both classifiers, such as the ANN and NB, obtained an accuracy of 82.38

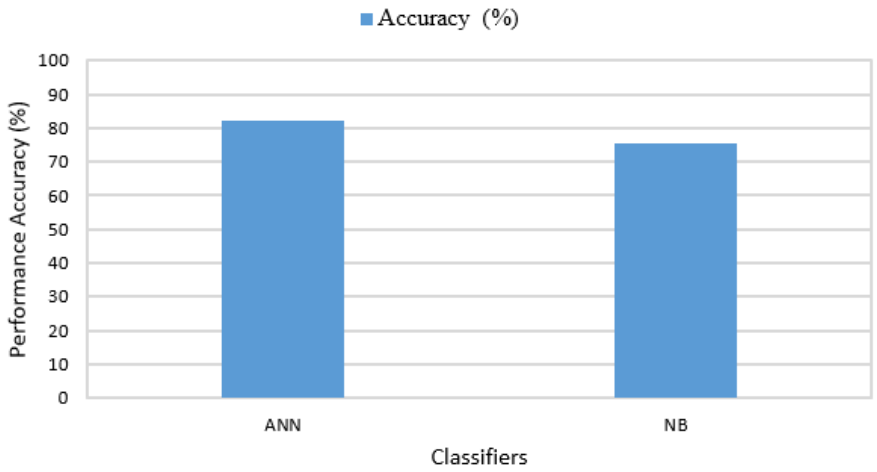


**Figure 3. Classification Results**

### *4.1. Confusion Matrix-based analysis of Results*

In this work, a confusion matrix (CF) based analysis is performed to analyze each implemented classifier's performance rate. A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but, more importantly, the types of errors that are being made. There are the following terms are used as listed below: **Positive (P):** Observation is positive (for example: is an apple).
**Negative (N):** Observation is not positive (for example, it is not an apple).
**True Positive (TP):** Observation is positive and is predicted to be positive.
**False Negative (FN):** Observation is positive but is predicted negative.
**True Negative (TN):** Observation is negative and is predicted to be negative.
**False Positive (FP):** Observation is negative but is predicted positive.
The following Table 3 is used to summarizes the results obtained using the ANN classifier. It can be observed that the $T_p$ rate of each class is higher as compared to other terms. The correct classification rate of the ANN classifier is 82.32%.

**Table 3. Confusion Matrix for ANN Classifier.**

| Predicted | | | | | |
|---|---|---|---|---|---|
| | **Text** | **Runtime** | **Resources** | **Releng** | **IDE** |
| **Class** | | | | | |
| **Text** | **1247** | 33 | 39 | 39 | 37 |
| **Runtime** | 47 | **826** | 15 | 21 | 55 |
| **Resources** | 30 | 31 | **676** | 27 | 33 |
| **Releng** | 41 | 48 | 42 | **1252** | 49 |
| **IDE** | 53 | 56 | 29 | 59 | **1113** |

The following Table 4 is used to summarizes the results obtained using the NB classifier. It is found that $T_p$ Rate of each class is higher as compared to other terms. In Table 4, it can be shown that the NB classifier correctly classified 4449 total number of instances out of 5998.

**Table 4. Confusion Matrix for Naive Bayes Classifier.**

| Predicted | | | | | |
|---|---|---|---|---|---|
| | **Text** | **Runtime** | **Resources** | **Releng** | **IDE** |
| **Class** | | | | | |
| **Text** | 1097 | 58 | 64 | 89 | 87 |
| **Runtime** | 72 | 726 | 45 | 51 | 70 |
| **Resources** | 64 | 65 | 553 | 50 | 65 |
| **Releng** | 83 | 94 | 74 | 1098 | 83 |
| **IDE** | 89 | 90 | 75 | 81 | 975 |

### *4.2. Performance Measures*

There are several performance measures available in the field of ML. These performance measures are used to evaluate the implemented classifiers to find the best classifier among all implemented classifiers. In this work, we used the following ML performance measures, such as accuracy (AC), recall (RE), Precision (PR), and F-measure (FM), to evaluate the performance of each classifier. The AC measure is used to validate the performance of the classifiers. It is a beneficial measure for the balanced dataset. It is the ratio of correct predictions to total predictions made. The following Equation 1 is used to calculate the AC ratio:

$$AC = \frac{T_p + T_n}{T_p + F_p + F_n + T_n} \tag{1}$$

However, there are problems with accuracy. It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor, or terrible depending upon the problem. Recall (RE) can be defined as the total number of correctly classified positive examples divided by the total number of positive examples. High Recall indicates the class is correctly recognized (a small number of $T_n$). The RE is defined as follows in Equation 2.

$$RE = \frac{T_p}{T_p + F_n} \tag{2}$$

To get the value of Precision (PR), we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example of labeled as positive is indeed positive (a small number of $F_p$). PR is defined as follows in Equation 3.

$$PR = \frac{T_p}{T_p + F_p} \tag{3}$$

High recall, low Precision: This means that most of the positive examples are correctly recognized (low FN), but there are a lot of false positives. Low recall, high Precision: This shows that we miss a lot of positive examples (high FN), but those we predict as positive are indeed positive (low FP). The F-measure (FM) is also defined as it uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more as expressed in Equation 4. The F-Measure (FM) will always be nearer to the smaller value of PE or RE.

$$FM = \frac{2 \times (PR \times RE)}{PR + RE} \tag{4}$$

Table 5 is used to summarize the experimental results obtained using the ANN and NB classifiers. It can be observed that the ANN classifier has a high accuracy rate as compared to the NB classifier. We observed that the ANN classifier has the highest classification results in terms of AC and recall values, and its accuracy is 94.55

**Table 5. Experimentation results**

| Classifier | AC (%) | RE (%) | PR (%) | FM (%) |
|---|---|---|---|---|
| **ANN** | 82.38 | 82.31 | 82.32 | 82.31 |
| **NB** | 75.43 | 75.41 | 75.40 | 75.40 |

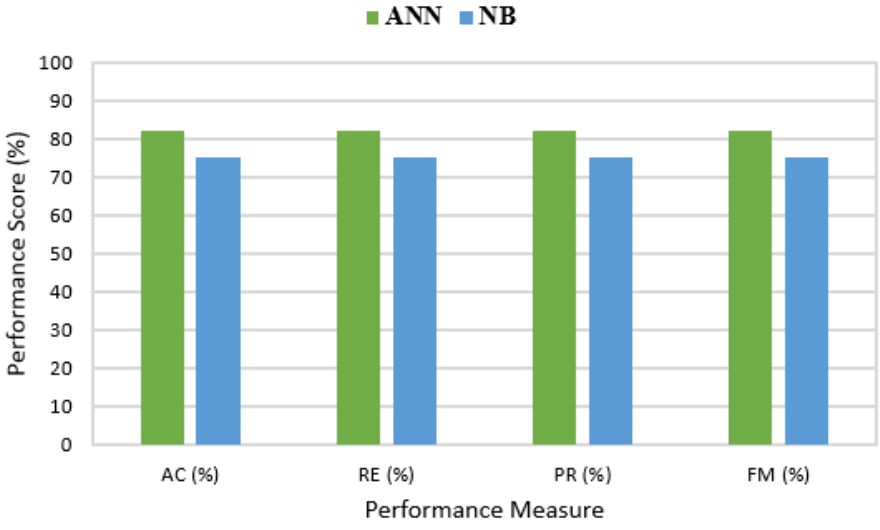


**Figure 4. Comparison of experimental results**

Total number of features: 18 Included in the classification process: 15

**Table 6. Data Distribution Summary**

| Bug Type | Total Number of Instances |
|---|---|
| **Text** | 1350 |
| **Runtime** | 914 |
| **Resources** | 997 |
| **Releng** | 1377 |
| **IDE** | 1260 |

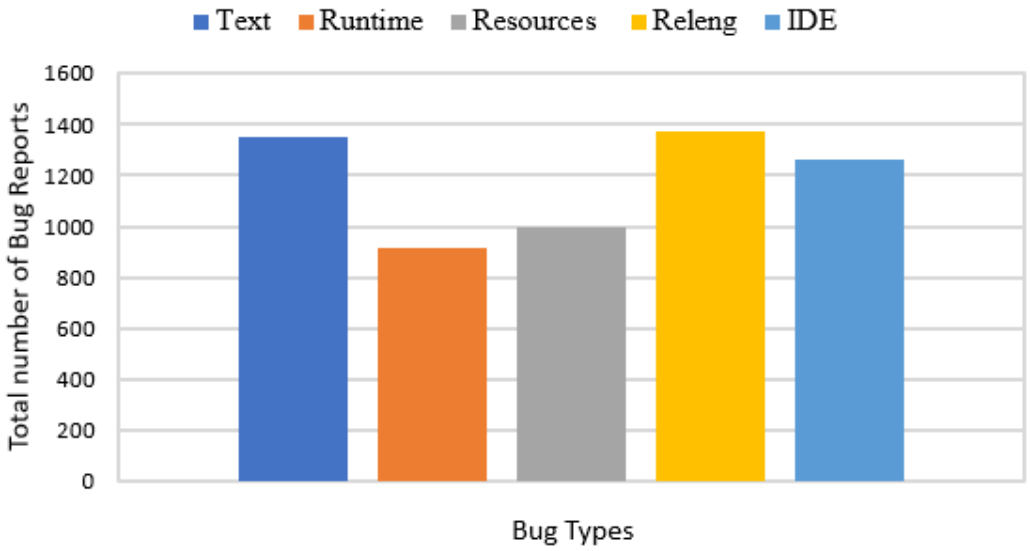


**Figure 5. Graphically Data Distribution.**

**Table 7. Proposed Features**

| # | Proposed Features |
|---|---|
| **Lexical Features** | |
| **1** | Words characters ratio |
| **2** | Total Number of Unique words |
| **3** | Unique words ratio |
| **4** | Total number of Stop words |
| **5** | Stop words ratio |
| **Syntactic Features (POS Tagging)** | |
| **6** | Total number of Nouns as Unigram |
| **7** | Total number of Verbs as Unigram |
| **8** | Total number of Adverbs as Unigram |
| **9** | Total number of Adjectives as Unigram |
| **10** | Total number of Interjections as Unigram |
| **Sentiment Features (Not Included)** | |
| **11** | Positive words ratio |
| **12** | Negative words ratio |
| **13** | Sentiment Score |
| **Statistical Features** | |
| **14** | Status of Bug i.e. resolved, verified, assigned etc. |
| **15** | Priority level of bug to solve soon (P1 to P5) |
| **16** | Operating system on which bug occurred |
| **17** | Severity of the occurred bug on the software system such as normal, major etc. |
| **18** | Total number of user reports. |

### *4.3. PREPROCESSING*

Pre-processing is a technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and lacking in certain behaviors or trends and is likely to contain many errors. Data pre-processing is a proven method of resolving such issues. In the Real-world data are generally incomplete: lacking attribute values, lacking specific attributes of interest, or containing only aggregate data.
**Noisy:** containing errors or outliers.
**Inconsistent:** containing discrepancies in codes or names.

### *4.4. Confusion matrix*

#### 4.4.1. Artificial neural networks classifier confusion matrix

Artificial neural networks (ANN), or connectionist systems, are computing systems inspired by, but not identical to, biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules. ANN Classification is the process of learning to separate samples into different classes by finding standard features between samples of known classes. ANN Classification is an example of Supervised Learning. Known class labels help indicate whether the system is performing correctly or not. This information can be used to indicate the desired response, validate the accuracy of the system, or be used to help the system learn to behave correctly. The known class labels can be thought of as supervising the learning process; the term is not meant to imply that you have some sort of interventionist role. The working of ANN takes its roots from the neural network residing in the human brain. ANN operates on something referred to as Hidden State. These hidden states are similar to neurons. Each of these hidden states is a transient form that has a probabilistic behavior. A grid of such a hidden state act as a bridge between the input and the output.

#### 4.4.2. Naive bayes classifier confusion matrix

Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong independence assumptions between the features. They are among the simplest Bayesian network models. Naïve Bayes classifiers are highly scalable, requiring several parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time rather than by expensive iterative approximation used for many other classifiers. A naive Bayes classifier is a model that assigns class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers but a family of algorithms based on a common principle. All naive Bayes classifiers assume that a particular feature's value is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features

## 5. Conclusion and future work

In this paper, we have proposed a technique to solve the bugs of the Bugzilla dataset. Whenever a bug comes in the Firefox or any other search engine, we relate it to the software development life cycle and relate it to one of the stages of the life cycle. And then, we suggest a solution for that bug. Whenever this bug occurs again, we will directly assign to that person oral solve directly to follow the previous steps, followed for the solution.

## References

1. Rodriguez, D.; Herraiz, I.; Harrison, R. On software engineering repositories and their open problems. 2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE). IEEE, 2012, pp. 52–56.
2. Liu, C.; Fei, L.; Yan, X.; Han, J.; Midkiff, S.P. Statistical debugging: A hypothesis testing-based approach. *IEEE Transactions on software engineering* **2006**, *32*, 831–848.
3. Voinea, L.; Telea, A. Mining software repositories with cvsgrab. Proceedings of the 2006 international workshop on Mining software repositories, 2006, pp. 167–168.
4. Śliwerski, J.; Zimmermann, T.; Zeller, A. When do changes induce fixes? *ACM sigsoft software engineering notes* **2005**, *30*, 1–5.
5. Shippey, T.; Bowes, D.; Hall, T. Automatically identifying code features for software defect prediction: Using ast n-grams. *Information and Software Technology* **2019**, *106*, 142–160.
6. Arshad, S.; Tjortjis, C. Clustering software metric values extracted from c# code for maintainability assessment. Proceedings of the 9th Hellenic Conference on Artificial Intelligence, 2016, pp. 1–4.
7. Tjortjis, C. Data Mining Code Clustering (DMCC): An approach supporting software maintenance and comprehension. Technical report, Technical report, School of Science & Technology, International Hellenic . . . , 2019.
8. Kanwal, J.; Basit, H.A.; Maqbool, O. Structural clones: An evolution perspective. 2018 IEEE 12th International Workshop on Software Clones (IWSC). IEEE, 2018, pp. 9–15.
9. Hindle, A.; German, D.M.; Holt, R. What do large commits tell us? A taxonomical study of large commits. Proceedings of the 2008 international working conference on Mining software repositories, 2008, pp. 99–108.
10. Tan, P.N.; Steinbach, M.; Kumar, V. Introduction to data mining, Pearson education. *Inc., New Delhi* **2006**.
11. Ekanayake, J.; Tappolet, J.; Gall, H.C.; Bernstein, A. Tracking concept drift of software projects using defect prediction quality. 2009 6th IEEE International Working Conference on Mining Software Repositories. IEEE, 2009, pp. 51–60.
12. Zimmermann, T.; Zeller, A.; Weissgerber, P.; Diehl, S. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering* **2005**, *31*, 429–445.
13. Fu, S.; Shen, B. Code bad smell detection through evolutionary data mining. 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE, 2015, pp. 1–9.
14. Naseem, R. An improved hierarchical clustering combination approach for software modularization. PhD thesis, Universiti Tun Hussein Onn Malaysia, 2017.
15. Schäfer, T.; Jonas, J.; Mezini, M. Mining framework usage changes from instantiation code. Proceedings of the 30th international conference on Software engineering, 2008, pp. 471–480.
16. Raza, U.; Tretter, M. Predicting software outcomes using data mining and text mining. SAS Global Forum, 2007.
17. Raghavan, S.; Rohana, R.; Leon, D.; Podgurski, A.; Augustine, V. Dex: A semantic-graph differencing tool for studying changes in large code bases. 20th IEEE International Conference on Software Maintenance, 2004. Proceedings. IEEE, 2004, pp. 188–197.
18. Rolfsnes, T.; Di Alesio, S.; Behjati, R.; Moonen, L.; Binkley, D.W. Generalizing the analysis of evolutionary coupling for software change impact analysis. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2016, Vol. 1, pp. 201–212.
19. German, D.M. An empirical study of fine-grained software modifications. *Empirical Software Engineering* **2006**, *11*, 369–393.